Summary

I have always been fascinated by fractals, as well as volumetric rendering. Seeing realistic volumetric clouds or even volumetric smoke effects within video games made me interested into learning about the techniques that go into rendering volumetric effects within Unreal Engine, and I chose fractals to be the centrepiece
Generating real time volumetric fractals in Unreal can be defined by two stages; first creating the fractal and then rendering it. The following documentation will use this differentiation to split up the methods presented.

Although there are many different and unique fractal sets, I wanted to use one that could be represented in a 3D way, to simplify the process. The Mandelbrot set is a fractal function where complex numbers do not diverge to infinity, meaning that one could zoom in on a point within the set and it would continue forever.
A three-dimensional version of this fractal has also been devised, constructed by Jules Ruis in 1997. This can technically not exist in 3D space, as there is no direct equivalent of two-dimensional complex numbers in three dimensions. However it is possible to create mandelbrot sets in the fourth-dimension using different numbering systems like quaternions and bicomplex numbers.
Many other forms of mandelbulb formulas exist, Quintic, Power-nine, spherical, cubic etc., but for the purpose of this project I will be using the main mandelbulb formula, by Daniel White and Paul Nylander, which uses spherical coordinates, which is an alternative coordinate system surmised of radial distance, polar angle and azimuthal angle.

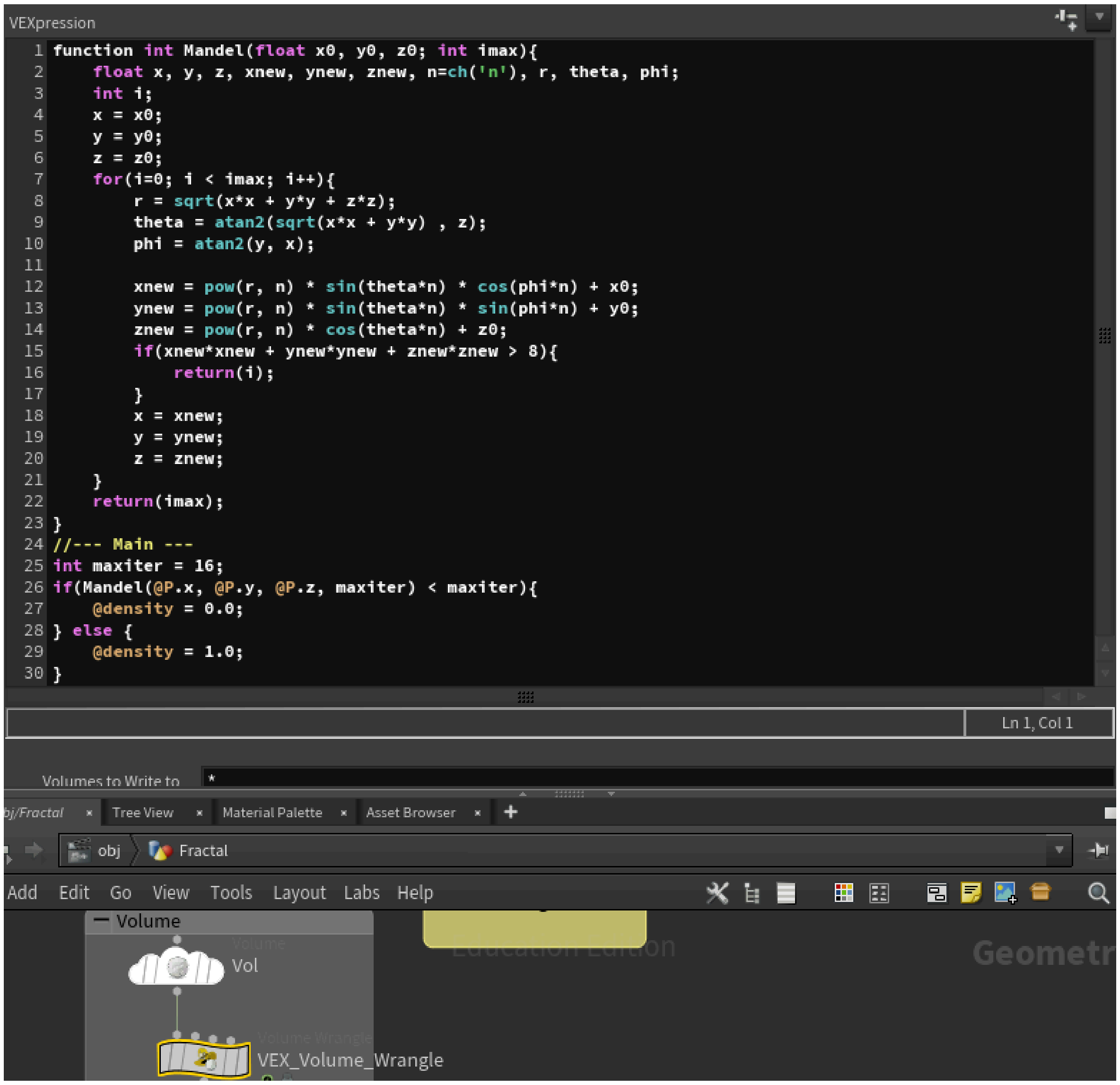
Goals

- Create Real-Time applications of fractals in Unreal using the mandelbrot/mandelbulb set.
- Understand the processes behind real-time volumetrics
- Understand the processes behind Raymarching within Unreal

Construction

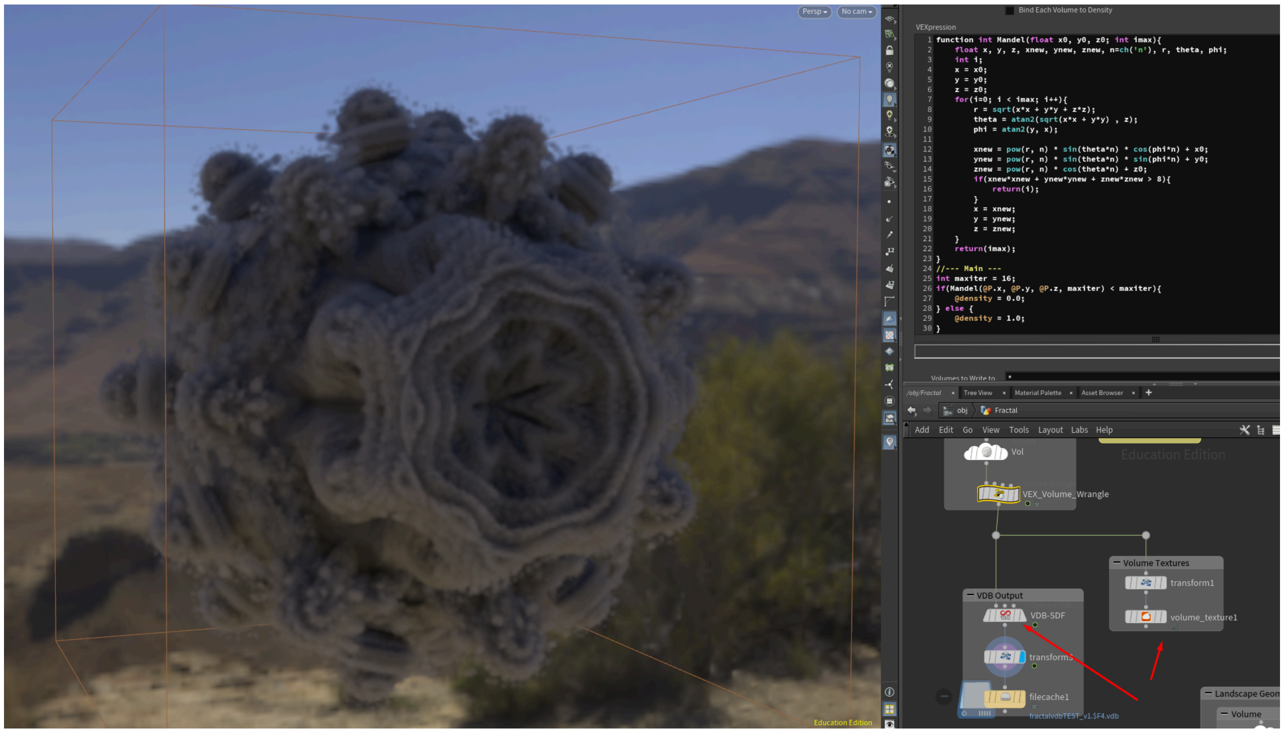
Houdini / VEX

Houdini's procedural generation toolset as well as its efficient node-based scripting workflow made it the most suitable candidate for creating fractals. Additionally Houdini allows quick changes to be made in real-time and much customisation into many workflows.
Most importantly, Houdini allows creation and manipulation of volumes and volumetric data.
After creating a volume, I was able to use a volume wrangle node to morph the volume into whatever shape I wanted. Using VEX (Vector EXpressions), Houdini's proprietary high-performance scripting language for writing functions into custom nodes and other various functions within Houdini.
The mandelbulb equation set is well documented online, and I was able to source a version converted for VEX use relatively quickly from Entagma, a source of Houdini knowledge online.



```
function int Mandel(float x0, y0, z0; int imax){
    float x, y, z, xnew, ynew, znew, n=ch('n'), r, theta, phi;
    int i;
    x = x0;
    y = y0;
    z = z0;
    for(i=0; i < imax; i++){
        r = sqrt(x*x + y*y + z*z);
        theta = atan2(sqrt(x*x + y*y) , z);
        phi = atan2(y, x);

        xnew = pow(r, n) * sin(theta*n) * cos(phi*n) + x0;
        ynew = pow(r, n) * sin(theta*n) * sin(phi*n) + y0;
        znew = pow(r, n) * cos(theta*n) + z0;
        if(xnew*xnew + ynew*ynew + znew*znew > 8){
            return(i);
        }
        x = xnew;
        y = ynew;
        z = znew;
    }
    return(imax);
}
//--- Main ---
int maxiter = 16;
if(Mandel(@P.x, @P.y, @P.z, maxiter) < maxiter){
    @density = 0.0;
} else {
    @density = 1.0;
}
```

From this I could then convert and output this volume in many forms; Volume Textures, VDB data, and polygonal data.

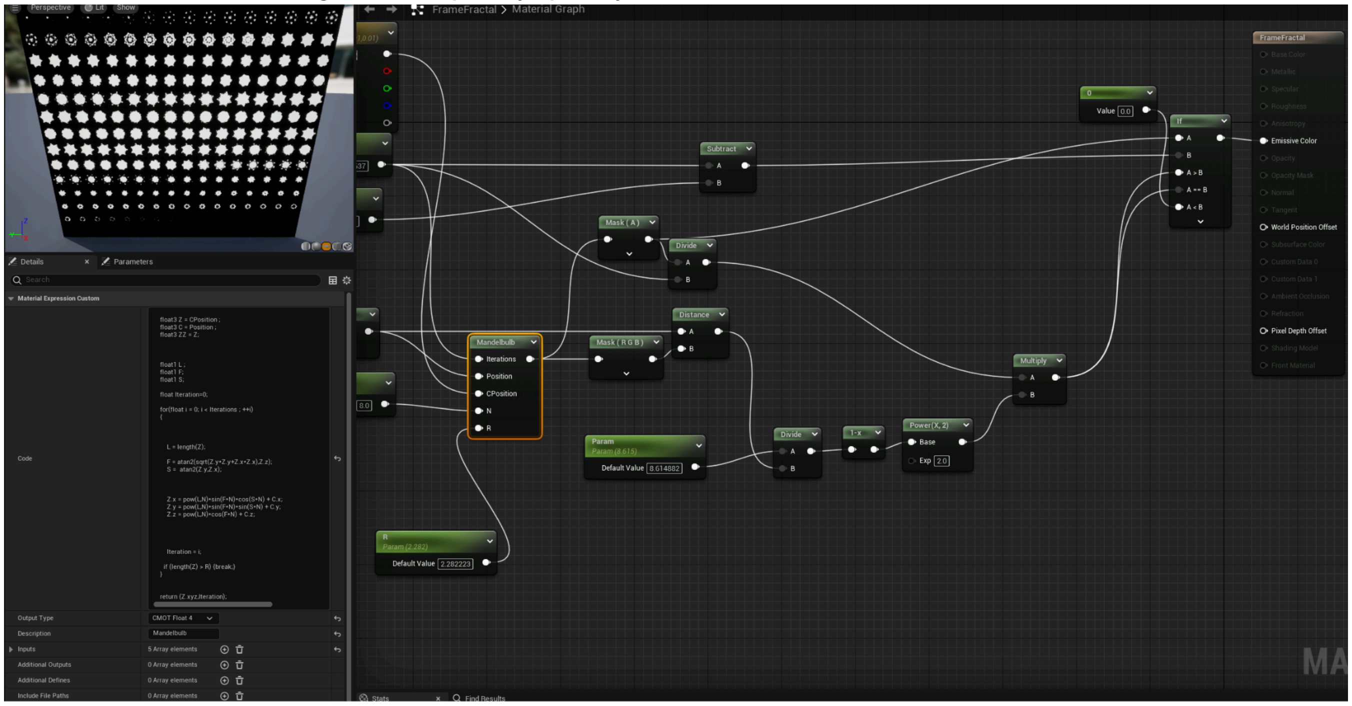


```
function int Mandel(float x0, y0, z0; int imax){
    float x, y, z, xnew, ynew, znew, n=ch('n'), r, theta, phi;
    int i;
    x = x0;
    y = y0;
    z = z0;
    for(i=0; i < imax; i++){
        r = sqrt(x*x + y*y + z*z);
        theta = atan2(sqrt(x*x + y*y) , z);
        phi = atan2(y, x);

        xnew = pow(r, n) * sin(theta*n) * cos(phi*n) + x0;
        ynew = pow(r, n) * sin(theta*n) * sin(phi*n) + y0;
        znew = pow(r, n) * cos(theta*n) + z0;
        if(xnew*xnew + ynew*ynew + znew*znew > 8){
            return(i);
        }
        x = xnew;
        y = ynew;
        z = znew;
    }
    return(imax);
}
//--- Main ---
int maxiter = 16;
if(Mandel(@P.x, @P.y, @P.z, maxiter) < maxiter){
    @density = 0.0;
} else {
    @density = 1.0;
}
```

I then imported these directly into Unreal-Engine, from which I could then move on to rendering these in different ways.

HLSL / Render Targets

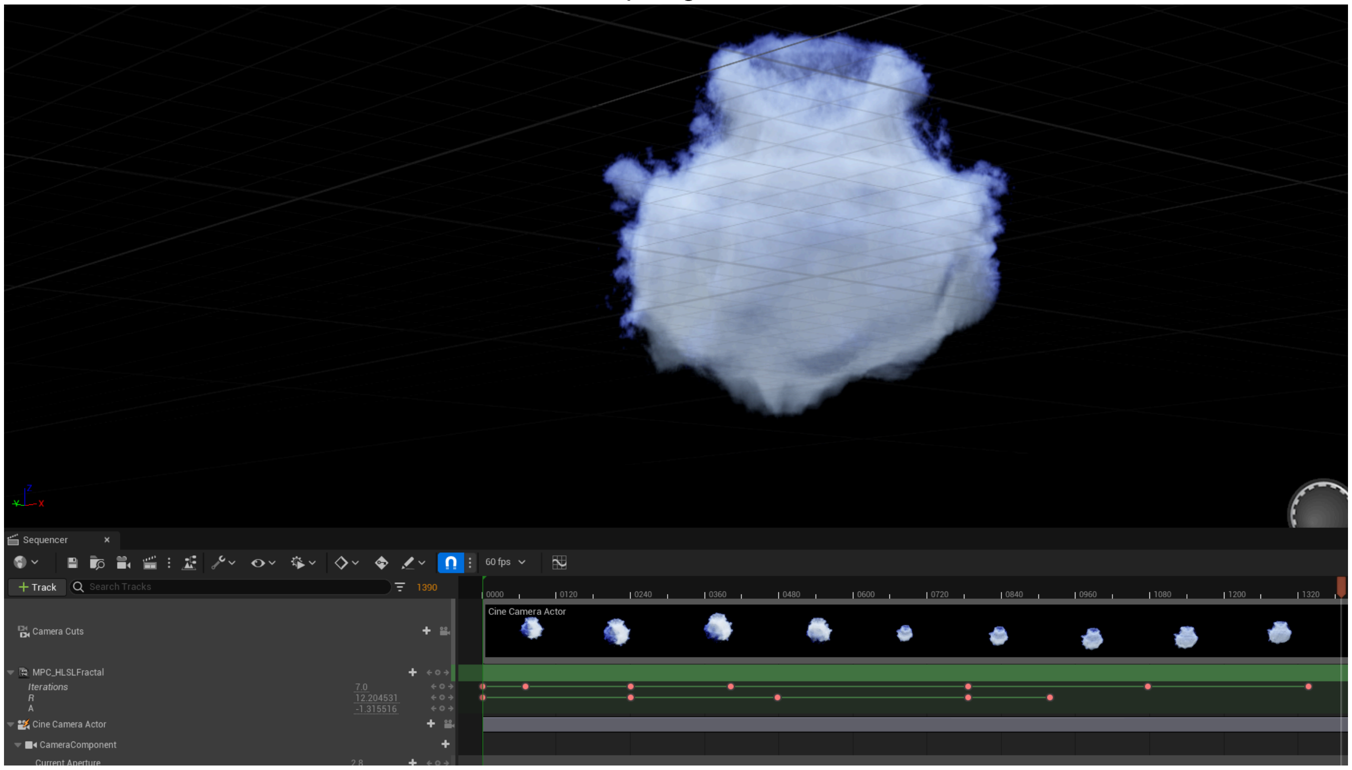
I was curious if it was possible to create a mandelbulb directly in Unreal Engine. It turns out that using HLSL, you can generate a mandelbulb within Unreal and convert it directly into a volume texture within the same shader. A technical artist online, by the name of Art Hiteca, demonstrated an implementation of this and I was able to get it set up very quickly.



```
float3 Z = CPosition ;
float3 C = Position ;
float3 ZZ = Z;



float1 L ;
float1 F;
float1 S;

float Iteration=0;

for(float i = 0; i < Iterations ; ++i)
{



   L = length(Z);

   F = atan2(sqrt(Z.y*Z.y+Z.x*Z.x),Z.z);
   S =  atan2(Z.y,Z.x);



   Z.x = pow(L,N)*sin(F*N)*cos(S*N) + C.x;
   Z.y = pow(L,N)*sin(F*N)*sin(S*N) + C.y;
   Z.z = pow(L,N)*cos(F*N) + C.z;




   Iteration = i;

  if (length(Z) > R) {break;}
}


return (Z.xyz,Iteration);
```

Output Type	CMOT Float 4
Description	Mandelbulb
Inputs	5 Array elements
Additional Outputs	0 Array elements
Additional Defines	0 Array elements
Include File Paths	0 Array elements

This render target texture can then be applied to a Density RayMarch (described in greater detail in the next section), by using a render target. The result is an average quality mandelbulb that can be manipulated in real-time by altering values on a material instance. I created an animated sequence where I animated various values to create a morphing mandelbulb.

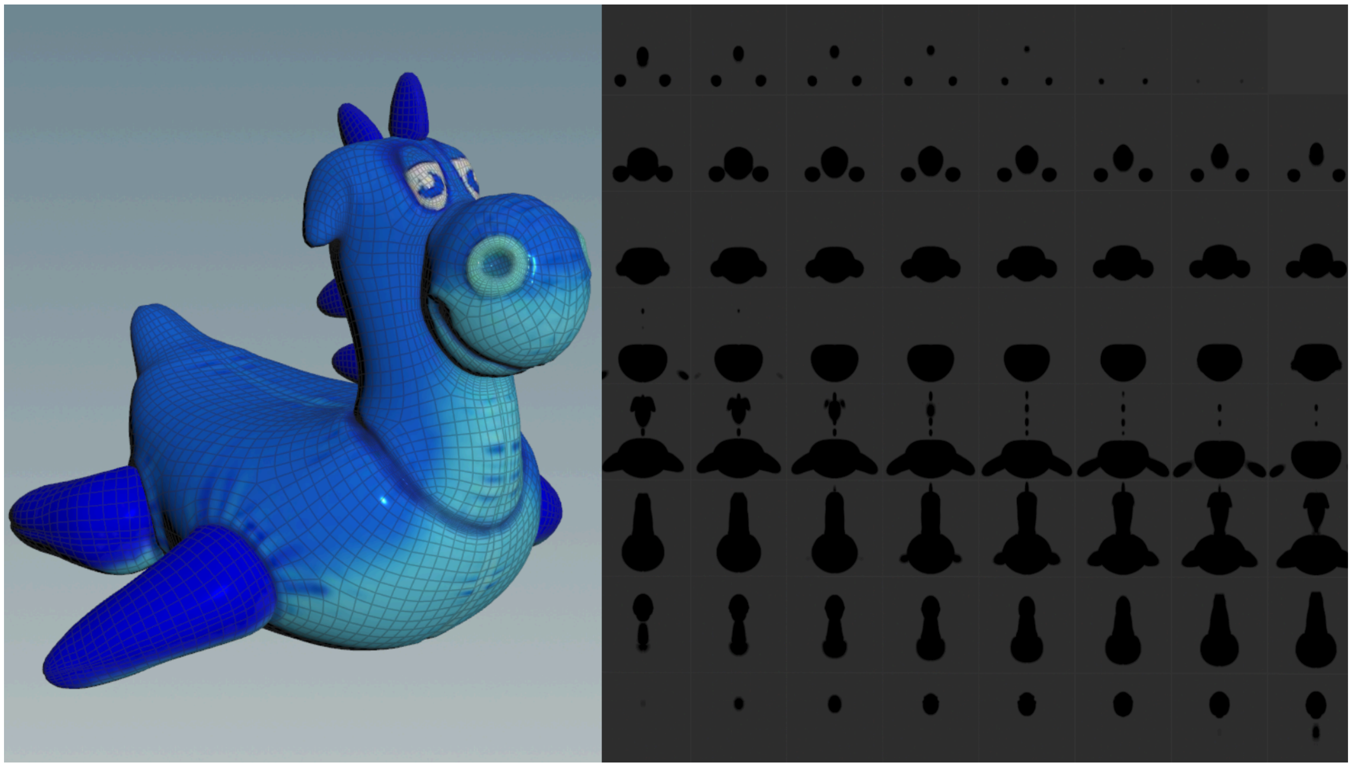


By no means does this method produce real-time capabilities whatsoever, drawing a render target for a volume texture is extremely budget consuming - as volume textures have resolutions in the many thousands, and updating this every frame makes it near impossible to perform well, and also makes it challenging to work with. For this reason I decided not to pursue this method further and focused more on pre-created mandelbulb data.

Rendering

Volume Textures

Although I would count volume textures as part of constructing, Volume textures are an obscure type of texture created by taking a three-dimensional shape and slicing it into cross sections.

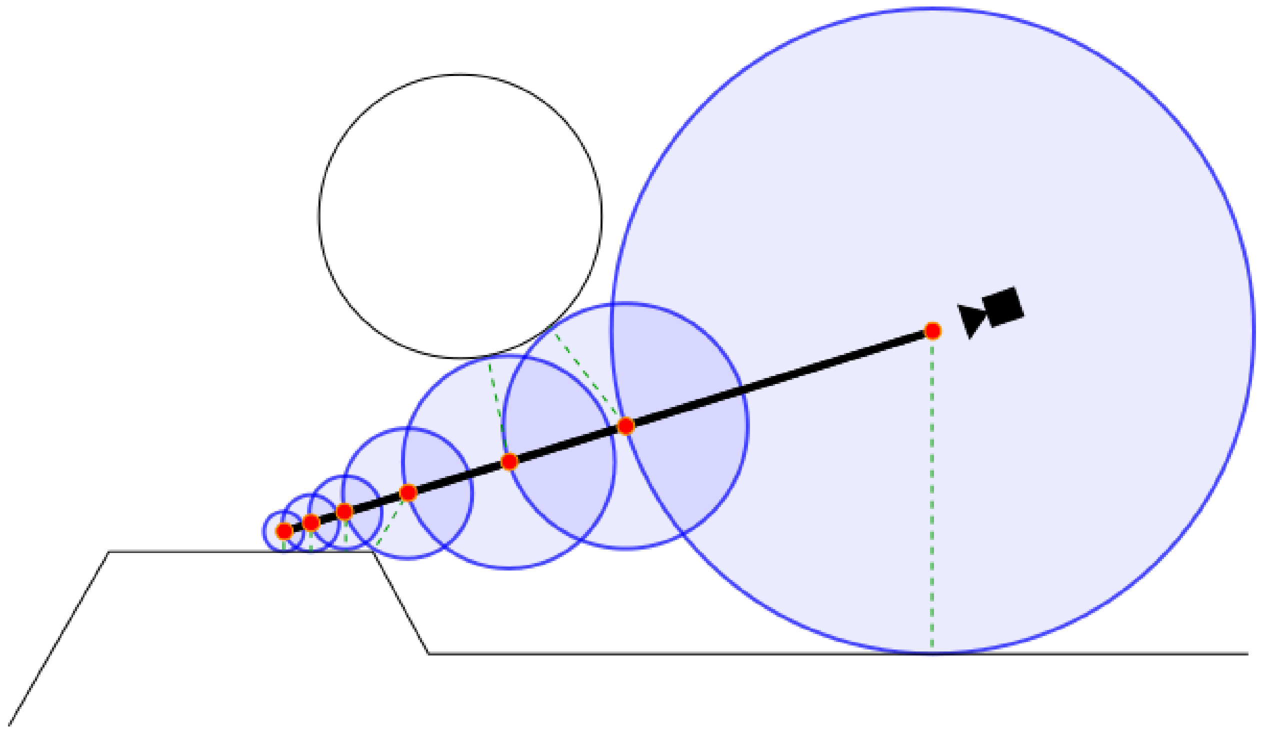

Test Geometry converted into a volume texture using Houdini

Volume textures have a multitude of uses, most commonly clouds and fluid simulations. It can allow complex fluid or cloud sequences to be baked into a texture for efficient real-time use.

Above test geometry’s volume texture converted into a real-time volume using a Density RayMarch.

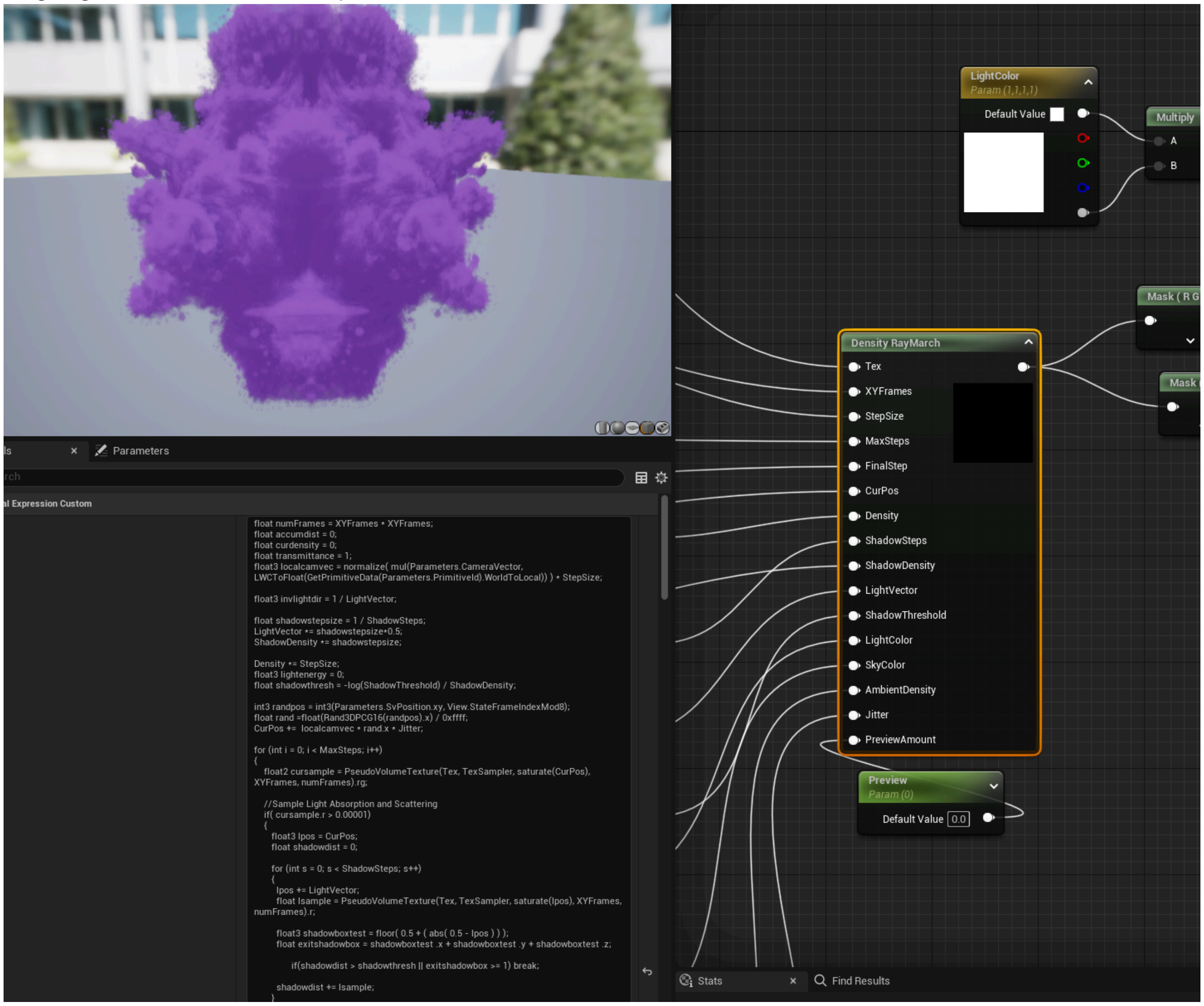
Ray Marching

Raymarching is a rendering technique used to render 3D scenes, as a variant of the more commonly used Ray Tracing, steps are taken along a ray, where the ray interacts with something within the scene, from the scene towards the camera, as opposed to drawing trace rays from the camera to the scene.



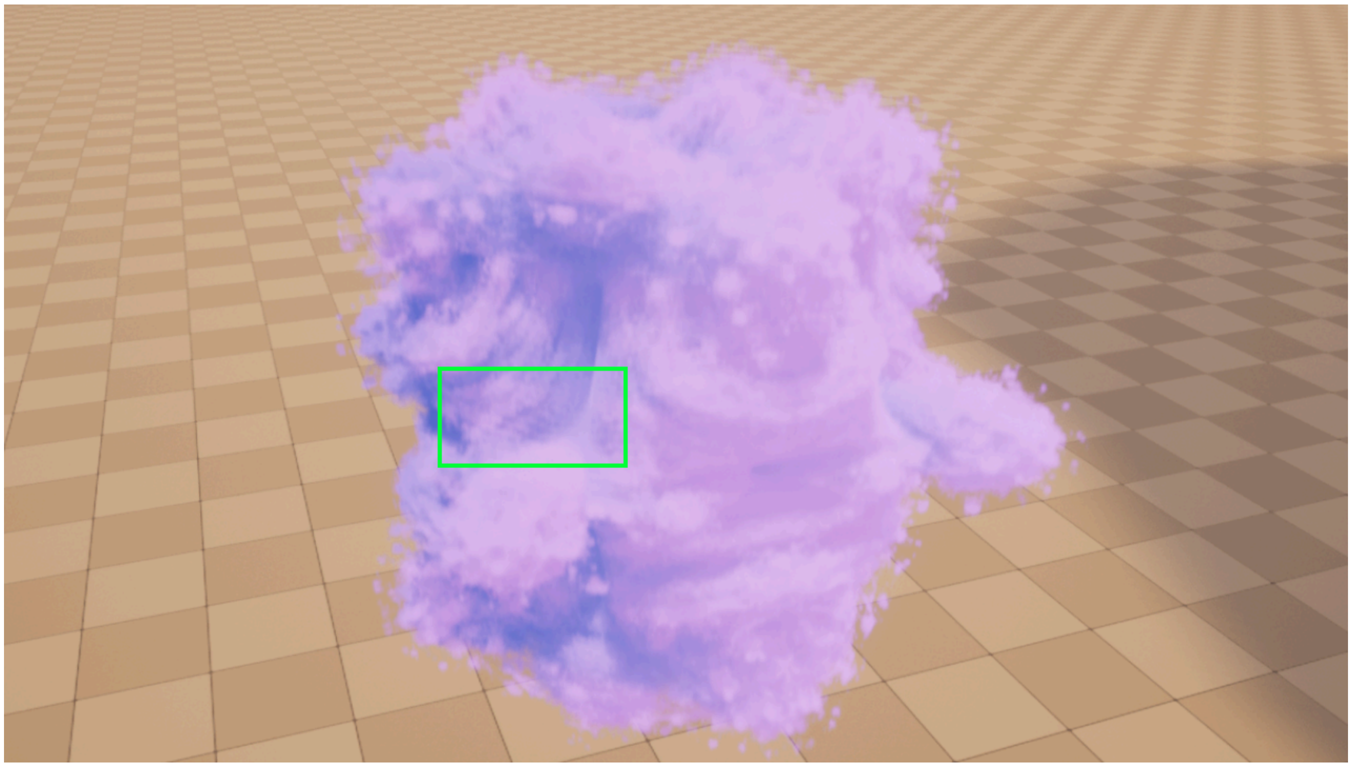
In Raymarching, we march along steps on the ray from the camera to as far as it can go straight away from the camera. At each step we check if it is intersecting with an object within the scene, if it does we calculate the colour at that point on the surface of the object. One of the more powerful applications of Raymarching is the rendering of volumetric data. By calculating the density at each data point, it can be determined how much light is absorbed at this particular point by the ray. There are many types of raymarching, and for this project a Density RayMarch will be used.

Raymarching in Unreal engine is not a straightforward process, as there is no easy access to a shader language, instead we must input HLSL code into a custom node for use within the material editor.



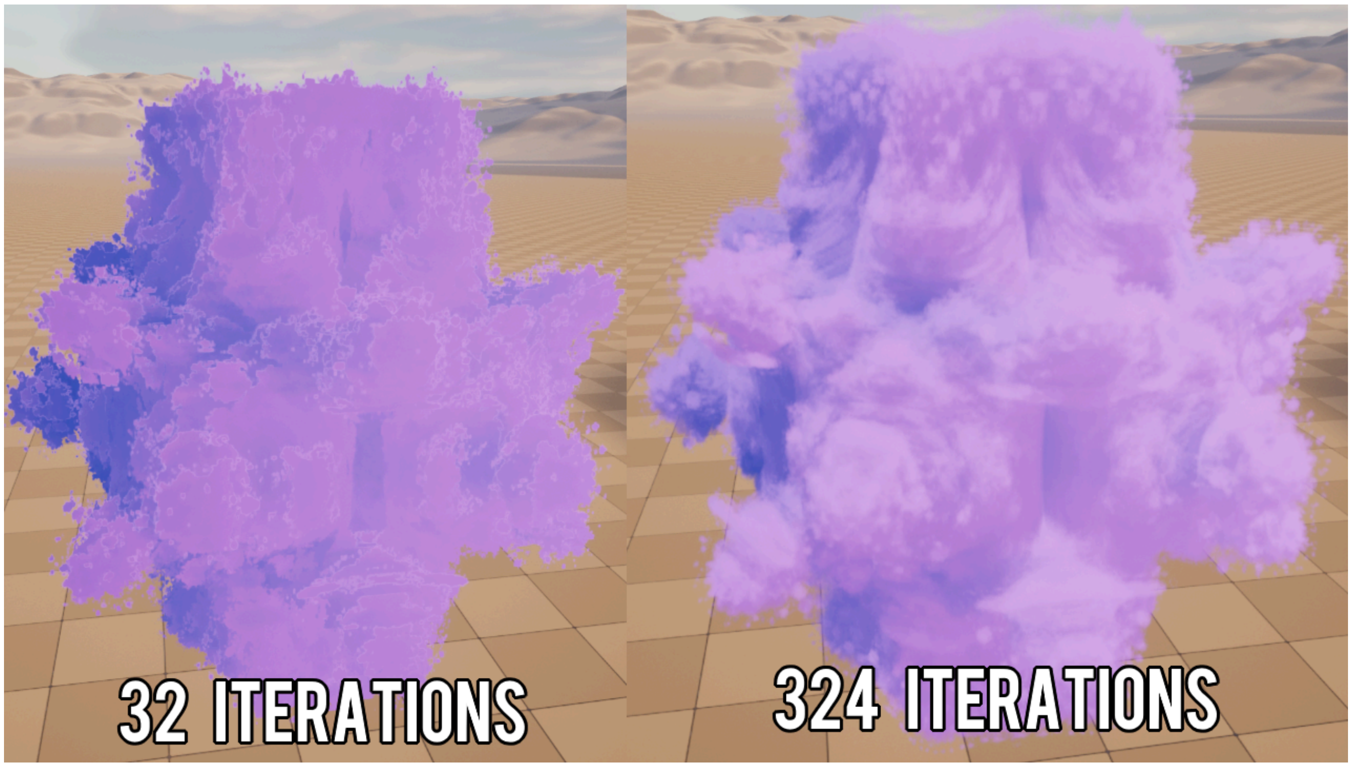
```
float numFrames = XYFrames * XYFrames;
float accumdist = 0;
float curdensity = 0;
float transmittance = 1;
float3 localcamvec = normalize( mul(Parameters.CameraVector,
LWCToFloat(GetPrimitiveData(Parameters.PrimitiveId).WorldToLocal)) ) * StepSize;

float3 invlightdir = 1 / LightVector;

float shadowstepsize = 1 / ShadowSteps;
LightVector *= shadowstepsize*0.5;
ShadowDensity *= shadowstepsize;

Density *= StepSize;
float3 lightenergy = 0;
float shadowthresh = -log(ShadowThreshold) / ShadowDensity;

int3 randpos = int3(Parameters.SvPosition.xy, View.StateFrameIndexMod8);
float rand =float(Rand3DPCG16(randpos).x) / 0xffff;
CurPos +=  localcamvec * rand.x * Jitter;

for (int i = 0; i < MaxSteps; i++)
{
   float2 cursample = PseudoVolumeTexture(Tex, TexSampler, saturate(CurPos),
XYFrames, numFrames).rg;

   //Sample Light Absorption and Scattering
   if( cursample.r > 0.00001)
   {
     float3 lpos = CurPos;
     float shadowdist = 0;

     for (int s = 0; s < ShadowSteps; s++)
     {
      lpos += LightVector;
      float lsample = PseudoVolumeTexture(Tex, TexSampler, saturate(lpos), XYFrames,
numFrames).r;

      float3 shadowboxtest = floor( 0.5 + ( abs( 0.5 - lpos ) ) );
      float exitshadowbox = shadowboxtest .x + shadowboxtest .y + shadowboxtest .z;

          if(shadowdist > shadowthresh || exitshadowbox >= 1) break;

      shadowdist += lsample;
     }
```

Luckily, many examples of raymarching nodes exist within the developer community, and I was able to find a plugin adapted for UE5 from a Game Developers Conference (GDC) talk from ShaderBits, a HLSL and UE4 shader Blog. They provided a custom node which allowed many inputs and modifications such as different directional lighting, density modifiers, iterations and other values which helped produce a nice looking result.



This looked great although there were some clear pixel artefacts due to the low X/Y slice count, as highlighted. I wanted to see how far I could go with the quality, so I went back to Houdini and increased the initial volume density, which will make the whole mandelbulb significantly more intricate and detailed after it has been wrangled. As well as this I stepped up the X/Y slice count from 11 to 22, and increased the resolution of each slice. I noticed immediately that increasing these values would result in decreased performance and any camera movements would cause the FPS to drop to under 30.
This made it clear that ultra high quality volume textures for ray marching would not be suitable for real time use, so I decided to do some cinematic renders to show a proof of concept of how it would work.

While ray marching produced a high quality result, it is one of the higher performance costing methods, as the higher the detail within the geometry (or volume), the more steps are needed to march along the ray to check for intersections. This is much more expensive on the CPU, as well as the fact that multiple iterations have to be taken to accurately calculate the density and colour at each point, as shown below.



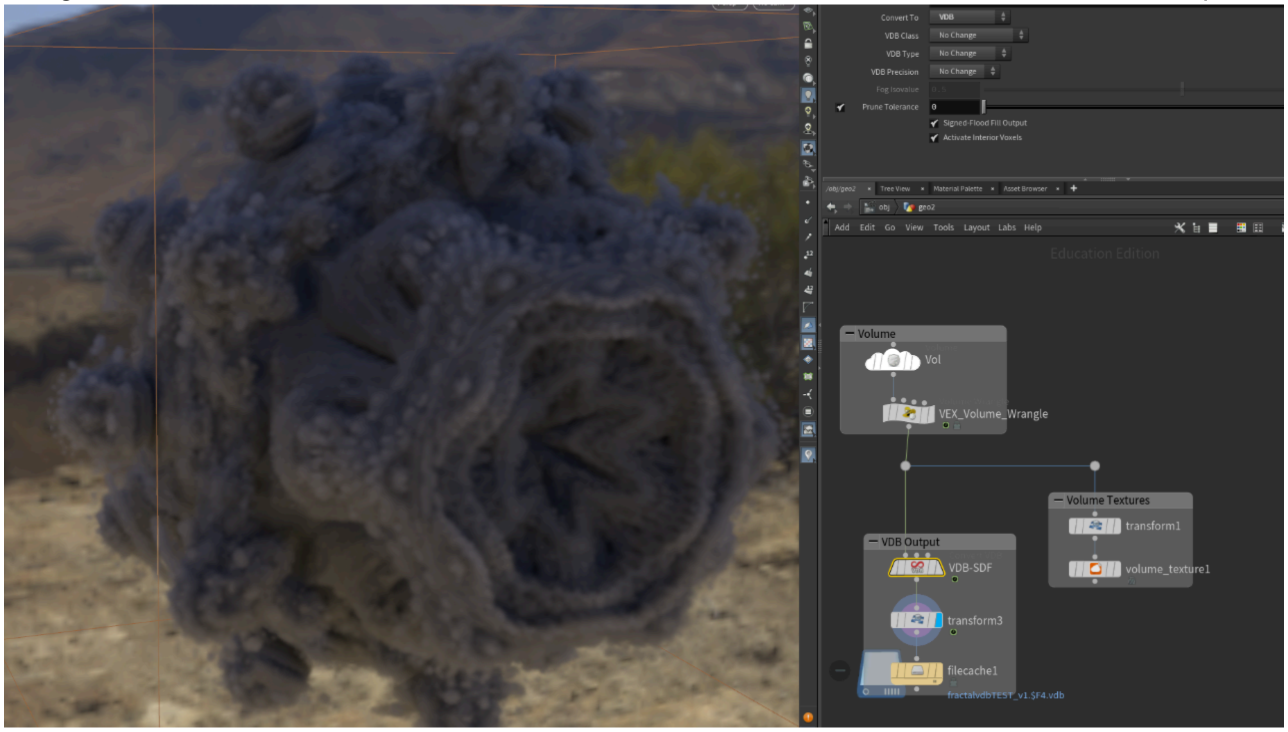
Higher iterations, as well as having to calculate light and colour values would result in much lower FPS while being rendered on the screen. It also has a more complex setup, requiring many material settings to be tuned in order to produce more optimised and cheap rendering. Therefore I found this method to not be suitable for real-time applications without a sacrifice of visual quality.

OpenVDB / NanoVDB

Volumetric Data can be stored in a format called *VDB* otherwise known as “Voxel Data Base”, or “Volumetric Data Blocks” as grid information in the form of scalar and vector values. OpenVDB is an award winning C++ library for efficient and optimised storage of sparse volumetric (VDB) data. Developed by Ken Museth and three other developers while working at DreamWorks Animation in 2012, it’s largely used for offline rendering, where performance is not a factor. However as rendering engines and hardware became more powerful, possibilities for real-time applications of VDB data arose.
NVIDIA took the opportunity to contribute real-time rendering GPU support for OpenVDB. It optimises the OpenVDB library by using a linearized, condensed and read-only version of the VDB tree structure. This means that all simulations can be baked into a VDB format, converted into NanoVDB and can be read and rendered in real time.
While NanoVDB has many industry partners who quickly adopted it such as Houdini, Blender or RENDERMAN, Unreal currently has had no native VDB support of any kind in any of its versions, it is also not listed on their roadmap despite many community requests for support to be added. Thus, this lead to community plugins such as SparseVolumetrics, developed by Thibault Lambert with the help of Sami Ibrahaim, which enables OpenVDB and NanoVDB support.

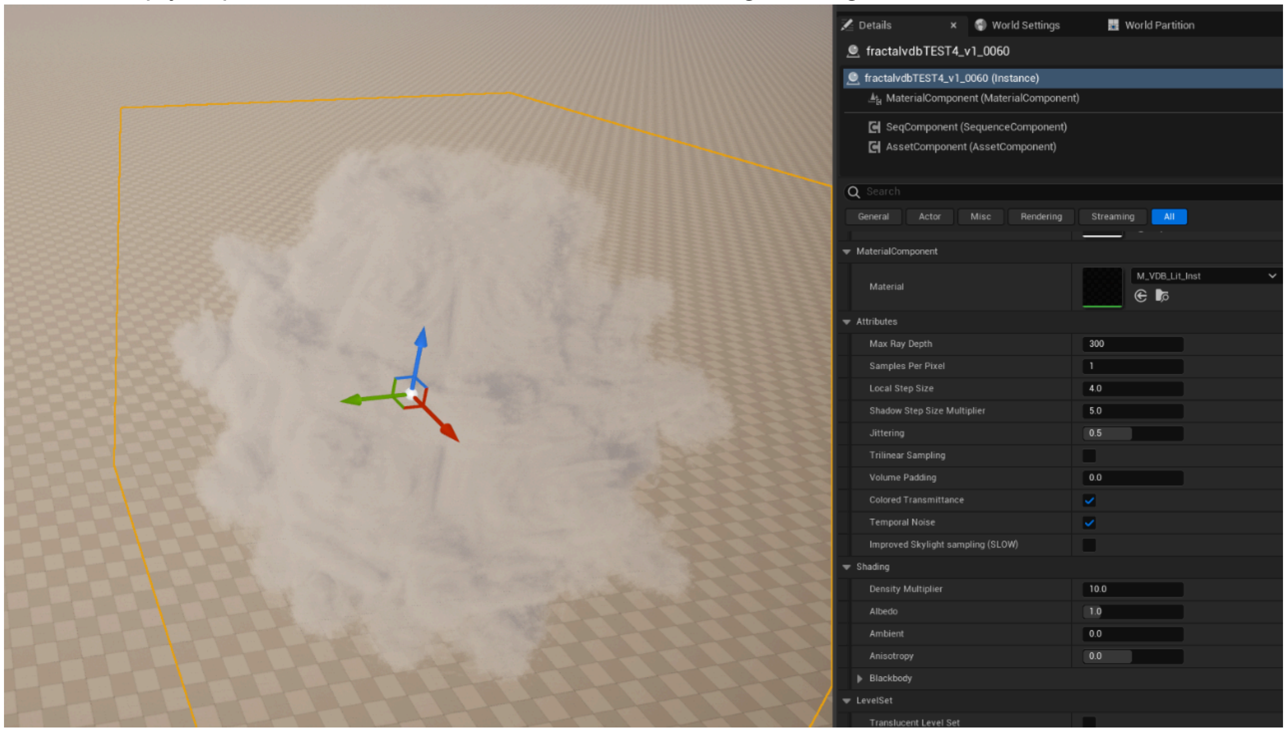
After installing this plugin I was surprised at how quick and painless they had made the process of importing VDB data. Simply dragging it into the content browser will allow importing of not only *sequences* of data meaning real-time animation, but also many forms of quantization, allowing VDB data to be compressed and therefore more lightweight.

Using the same volume from Houdini, I converted the volume to a VDB data structure and exported it.



Convert To	VDB
VDB Class	No Change
VDB Type	No Change
VDB Precision	No Change
Fog Isovalue	0.5
✓ Prune Tolerance	0
✓ Signed-Flood Fill Output	
✓ Activate Interior Voxels	

Then, I simply imported it into Unreal and was able to drag it straight into the world with no extra setup.

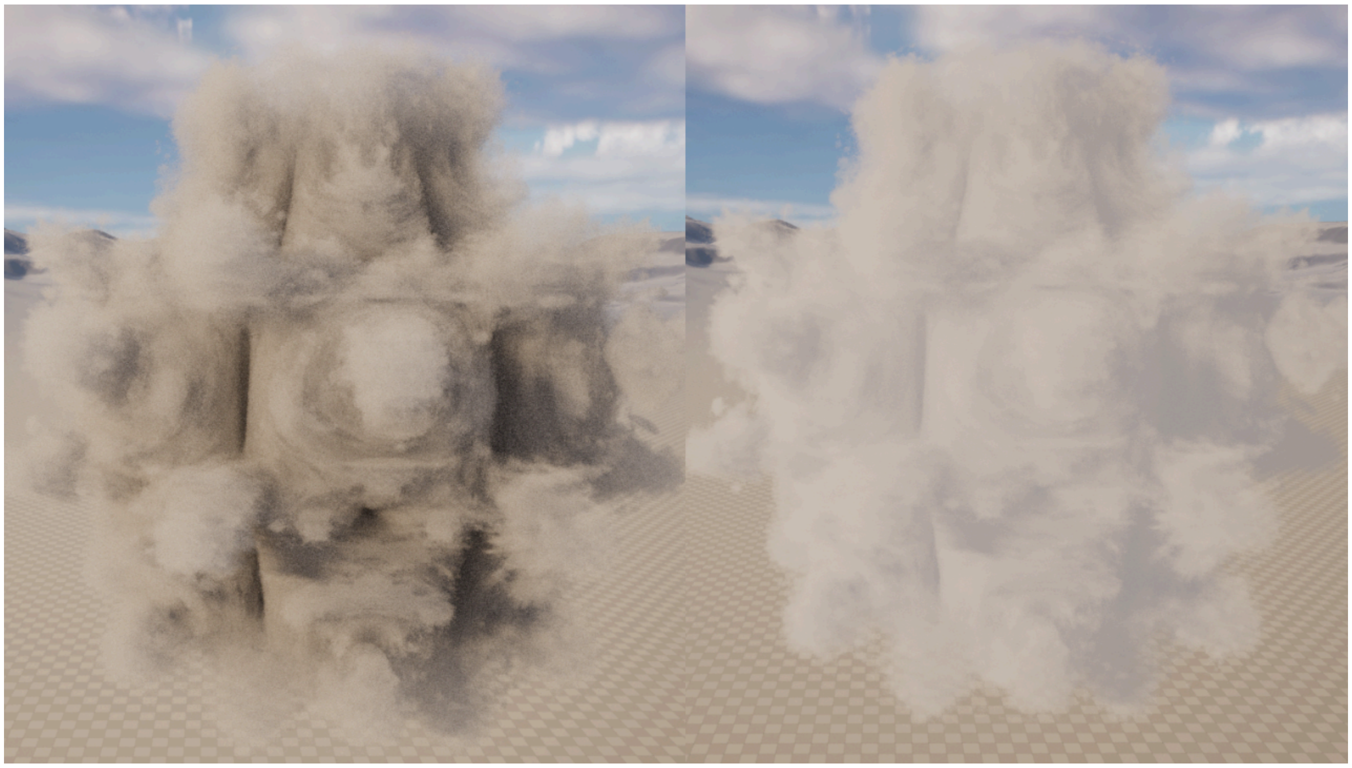


fractalvdbTEST4_v1_0060

fractalvdbTEST4_v1_0060 (Instance)
MaterialComponent (MaterialComponent)
SeqComponent (SequenceComponent)
AssetComponent (AssetComponent)

MaterialComponent	
Material	M_VDB_Lit_Inst
Attributes	
Max Ray Depth	300
Samples Per Pixel	1
Local Step Size	4.0
Shadow Step Size Multiplier	5.0
Jittering	0.5
Trilinear Sampling	☐
Volume Padding	0.0
Colored Transmittance	✓
Temporal Noise	✓
Improved Skylight sampling (SLOW)	☐
Shading	
Density Multiplier	10.0
Albedo	1.0
Ambient	0.0
Anisotropy	0.0
Blackbody	
LevelSet	
Translucent Level Set	☐

As shown, there are many rendering options, including an optional increased quality skylight sampling mode.

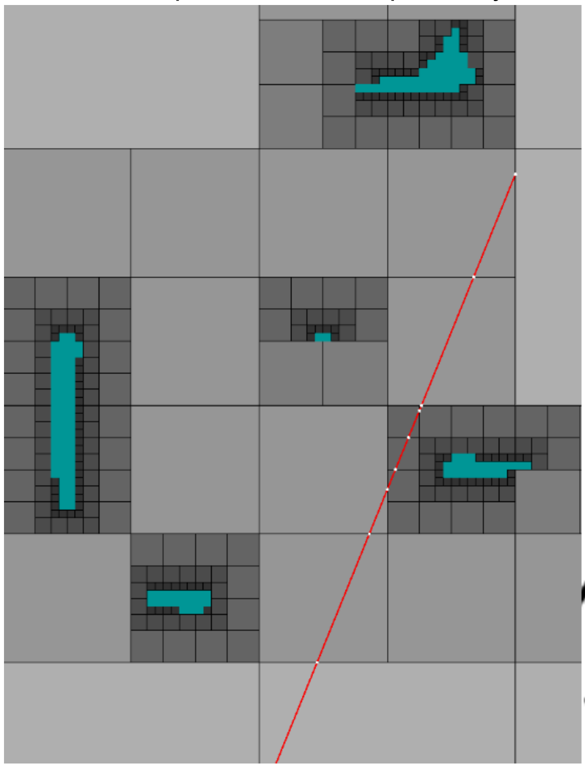


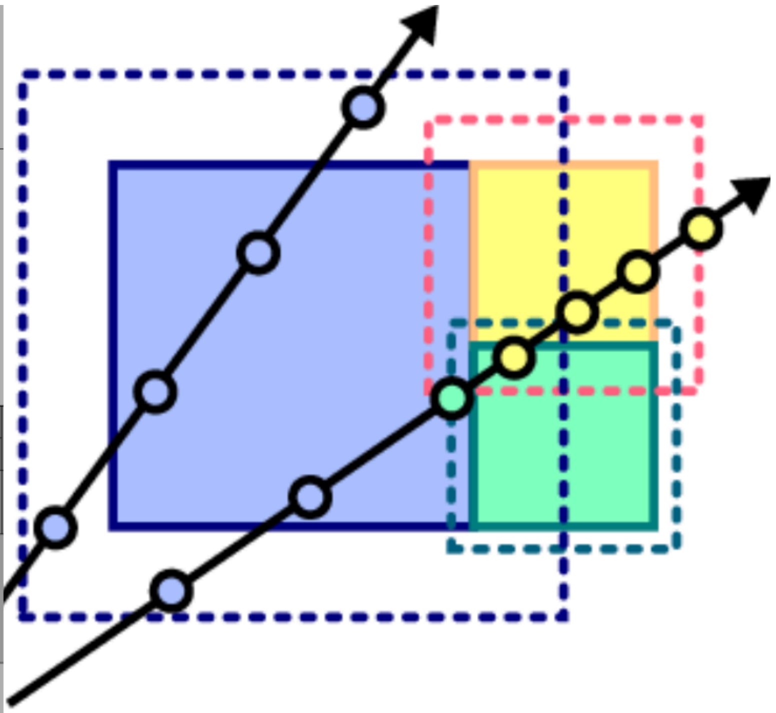
There is a noticeable difference in performance between these two.

Volumetrics

Unreal's volumetric cloud system was released with engine version 4.26 on the 10th December 2020. The new component allows users to create realistic and volumetric looking clouds for relatively low rendering cost. This also uses Raymarching, however it is much more efficient due to various optimisations developed by Epic Games;

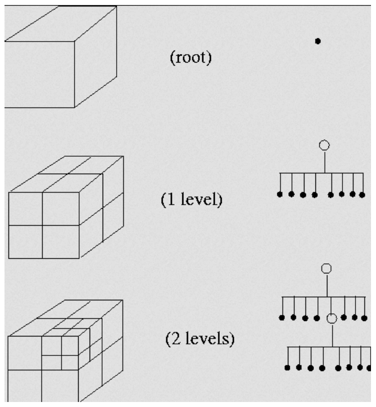
Adaptive sampling changes the amount of steps taken along the ray depending on the complexity of the cloud shape as well as its proximity towards the camera.



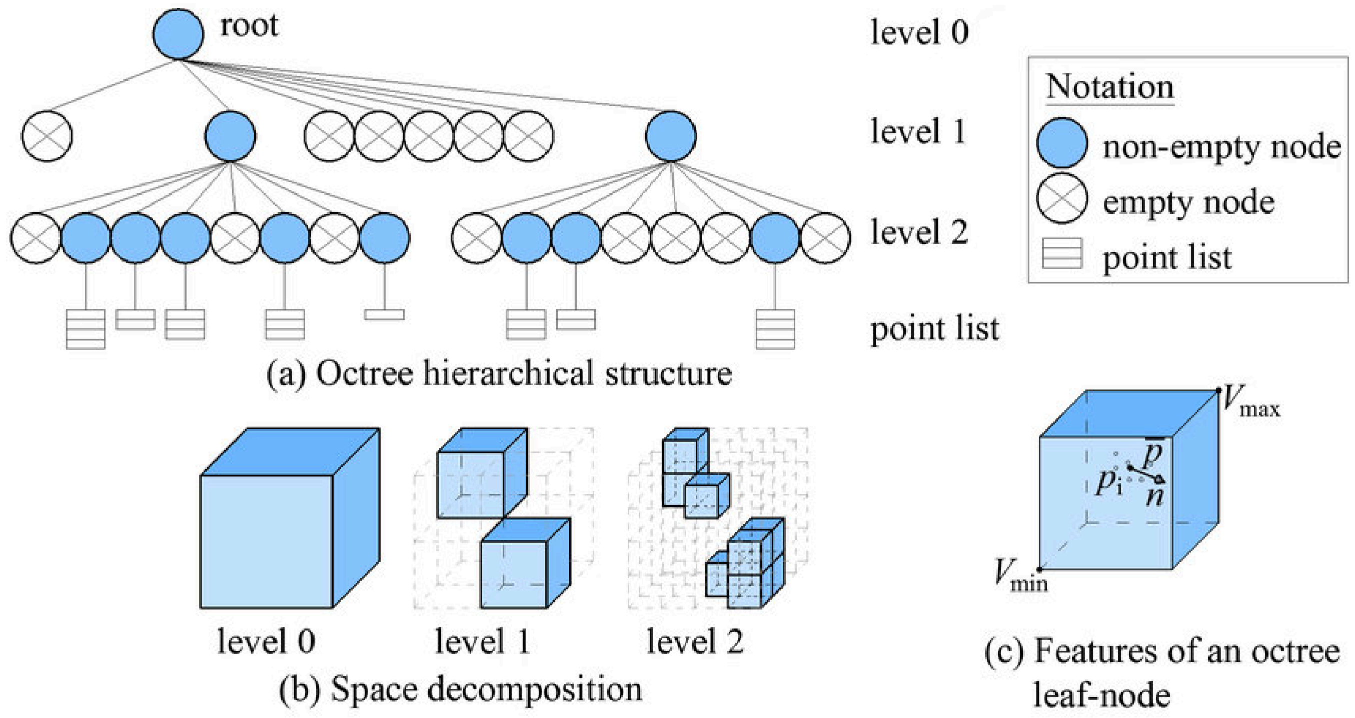


This reduces the amount of calculations done along the ray and therefore improves performance without affecting quality. As the rays are tested against boundaries of each region, it is possible to determine how many steps are needed at each sampling point. This can be applied to geometry/volumes where the detail of a shape as well as its distance from the camera determines the amount of sampling steps taken within this region. Due to its capabilities and customisation I wanted to be able to create cloud shapes using fractals.

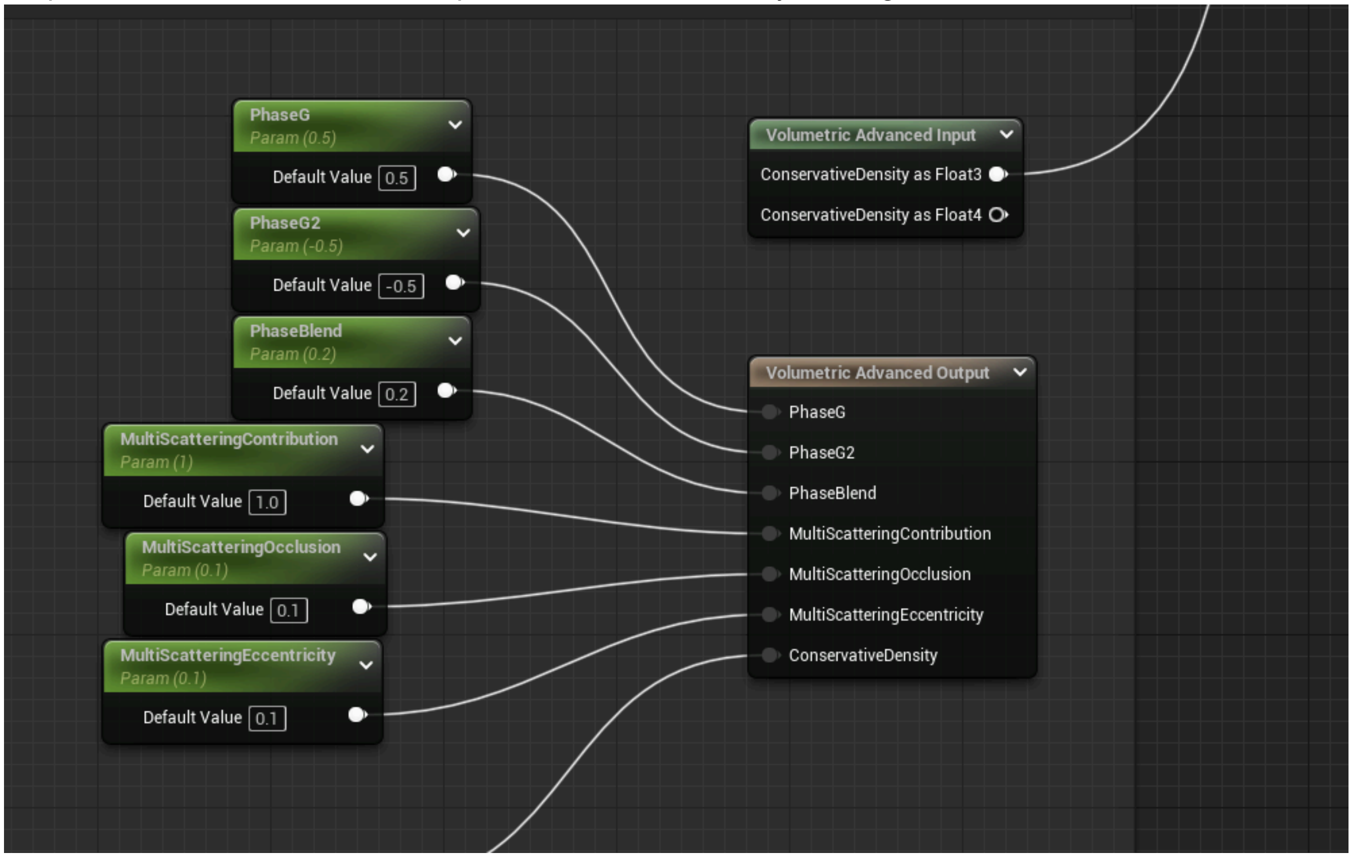
But, how do we get these "regions", well this is the secondary main technique that Unreal employs to optimise their volumetric raymarching. *Hierarchical data structures* such as *Octree,* developed by Donald Meagher in 1980, are a way to procedurally break up a three-dimensional space into smaller partitions, commonly referred to as 'cells'.



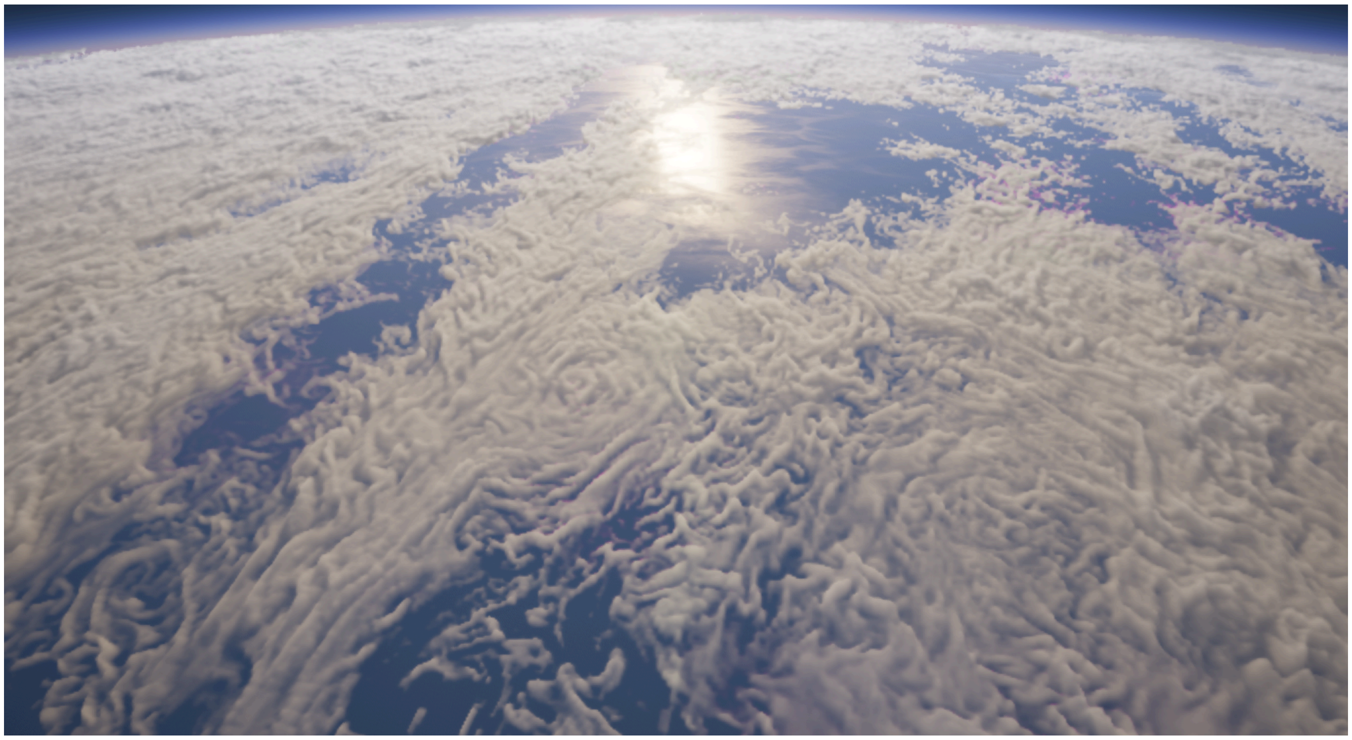
For example, if there is a cell that intersects with high detail or many objects, it makes more sense to subdivide the cell so that the ray only has to check the smaller cells that the ray intersects, rather than the whole cell with many objects.



Luckily, Unreal does all of this work behind the scenes, by feeding data into their 'Advanced Volumetric Output' node we can have all this optimisation done for us by the engine.

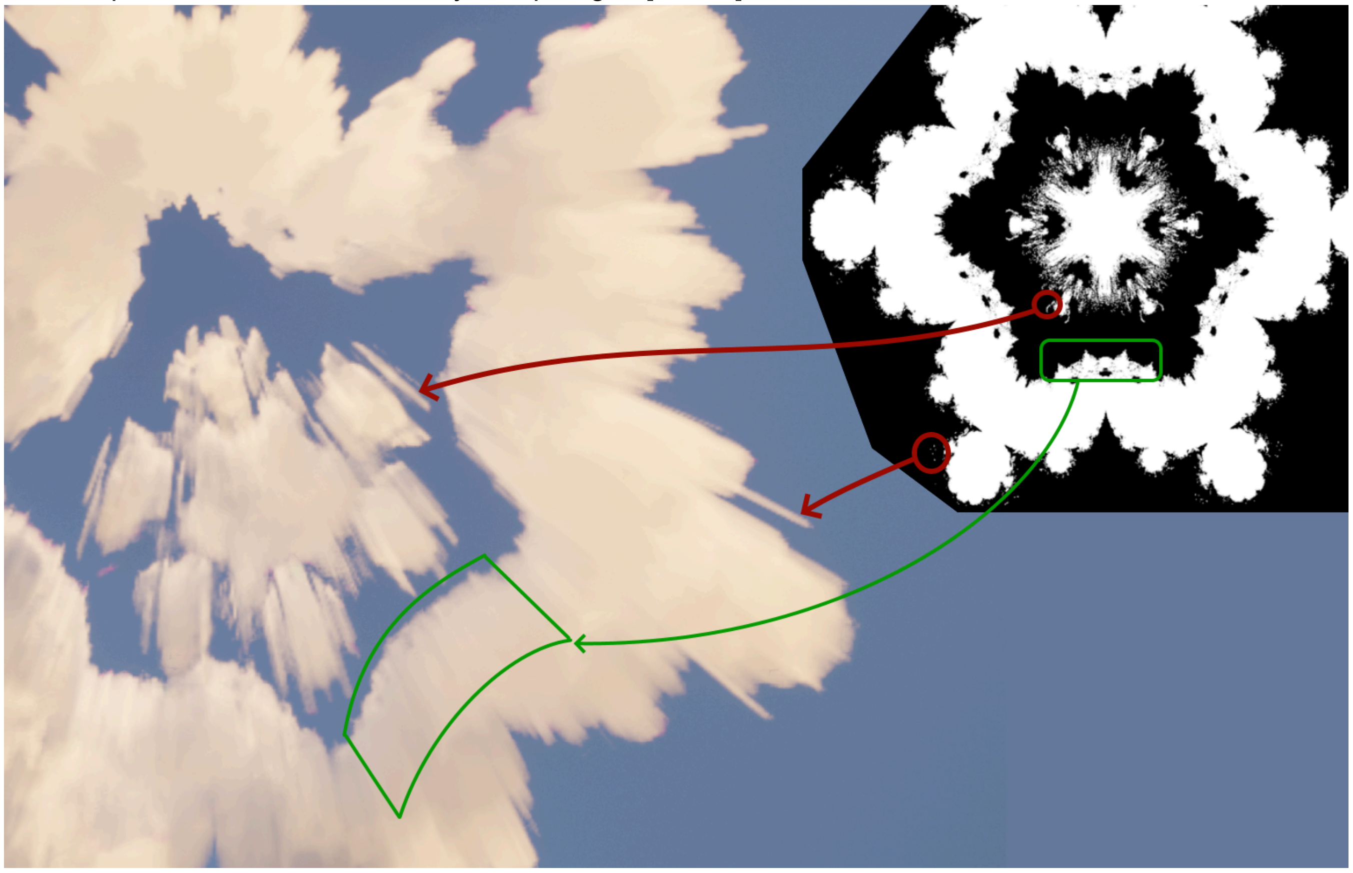


While Unreal has a very customisable volumetric cloud system by default, I wanted to have clouds that morphed through many different fractal shapes. So I looked towards a popular community plugin called FluidNinja - a baking toolkit for fluid and volumetric simulation.
They provide several use cases within their project of volume textures and flow maps used to create volumetric effects, as shown below.



I adapted one of their master materials which used many of their custom material functions called AtlasPlay - intended for flipbooking through volume textures for cloud materials. This works great as it allowed me to animate clouds in a way that wasn't just noise.

My initial tests with this method allowed me to use a simple volume texture of the mandelbulb to morph through cross-sections of its shape, however I noted that due to the volume texture having mostly black or white pixels - there would be very sharp edges [Green]

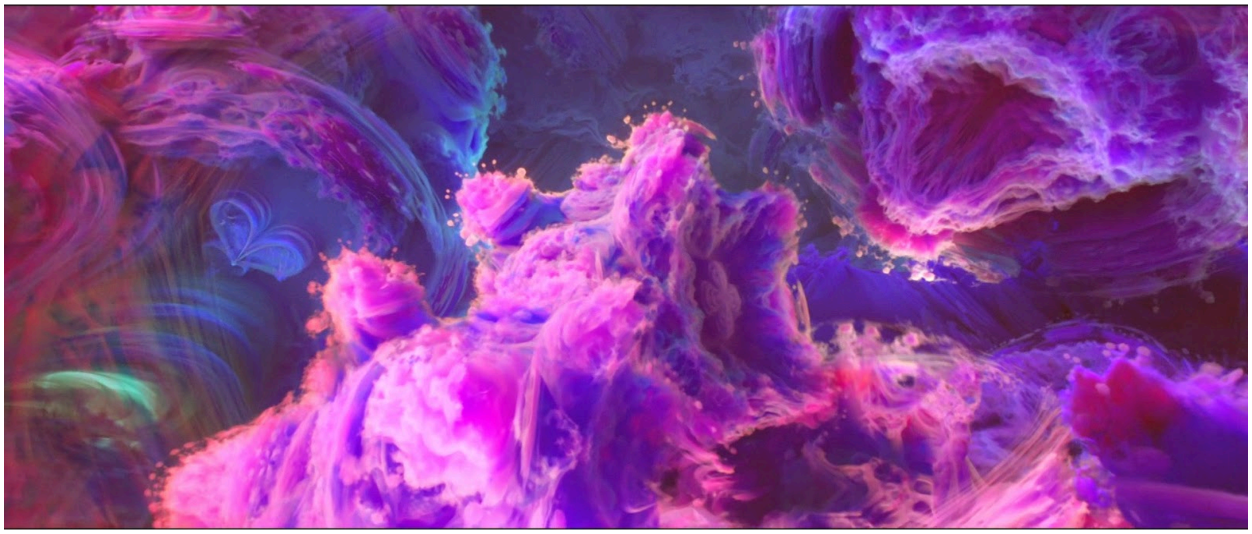


I also noted that single small island pixels would be as tall as more solid parts [Red]. I wanted to reduce the frequency of these artefacts and give the clouds an actual "shape" rather than looking like a wedge of clouds. The way to achieve this would be to have grayscale values representing depth.

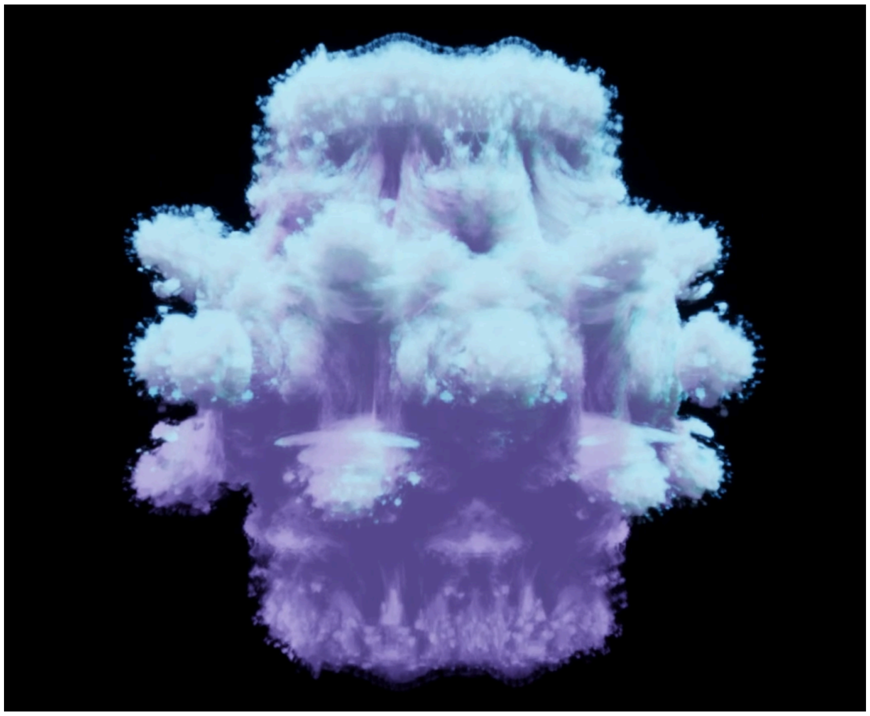
Creating Demo Scenes

Ray Marching

Inspired by one of the final scenes in Big Hero 6, by Disney Animation Studios, where the mandelbulb formula is used to create a vibrant environment for the inside of a wormhole.

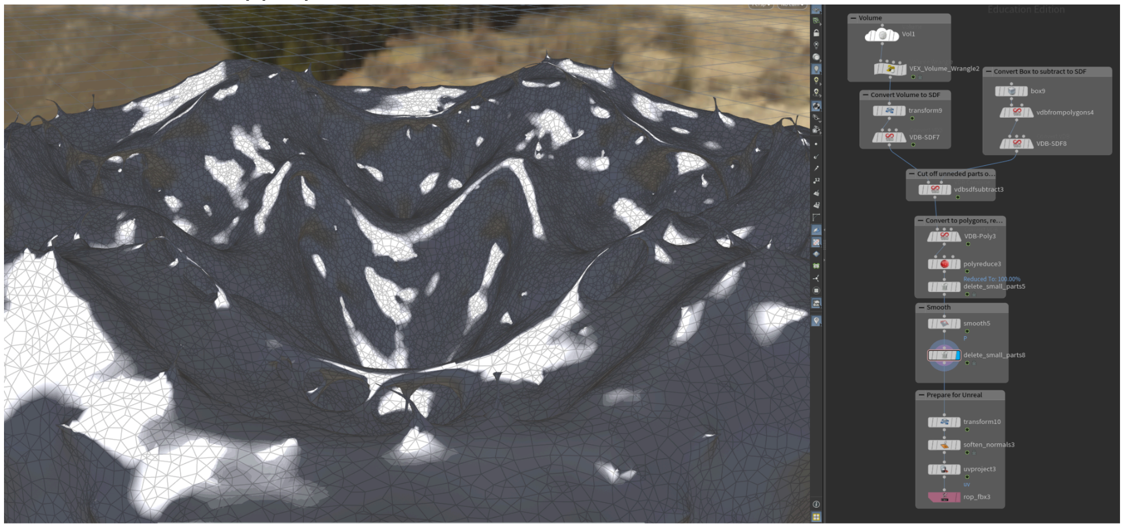


I loved the pink and purple look, and wanted to use my own spin on this idea. I chose purple with blue lighting to create a wispy colourful appearance. I was also able to animate the lights position within the sequencer to add additional flair to the lighting.

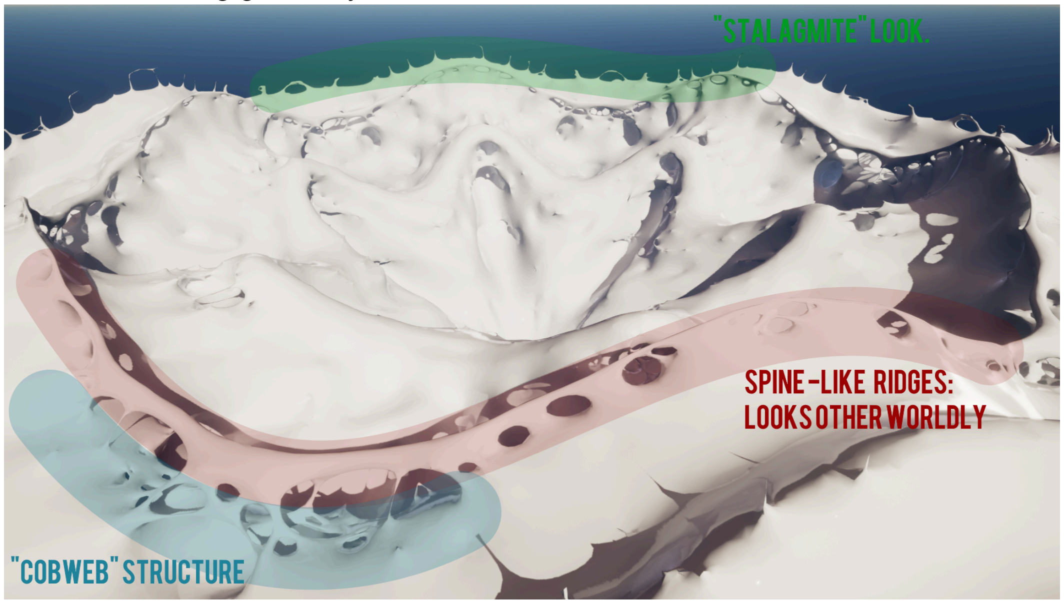


Volumetrics

For the scene that will contain these volumetric fractal clouds, I wanted to create a mountainous landscape using the mandelbulb volume.
So, utilising Houdini I converted the mandelbulb from a volume to polygons. I then used several additional nodes to reduce polygons, cut off sections I did not need, smooth much rough geometry, project the UVs face downward so that I could put landscape materials on it, soften normals, and transform it to an appropriate scale for Unreal.

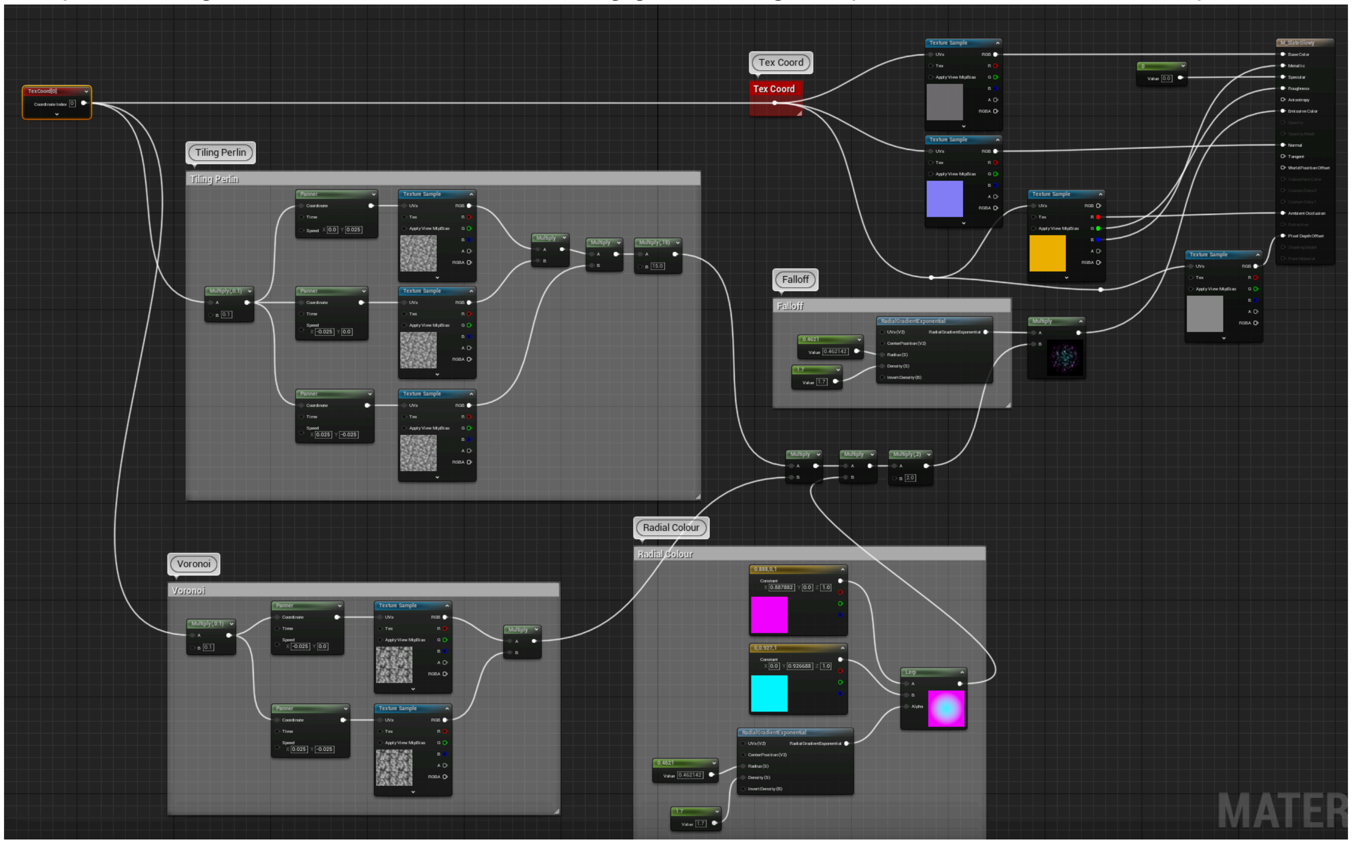


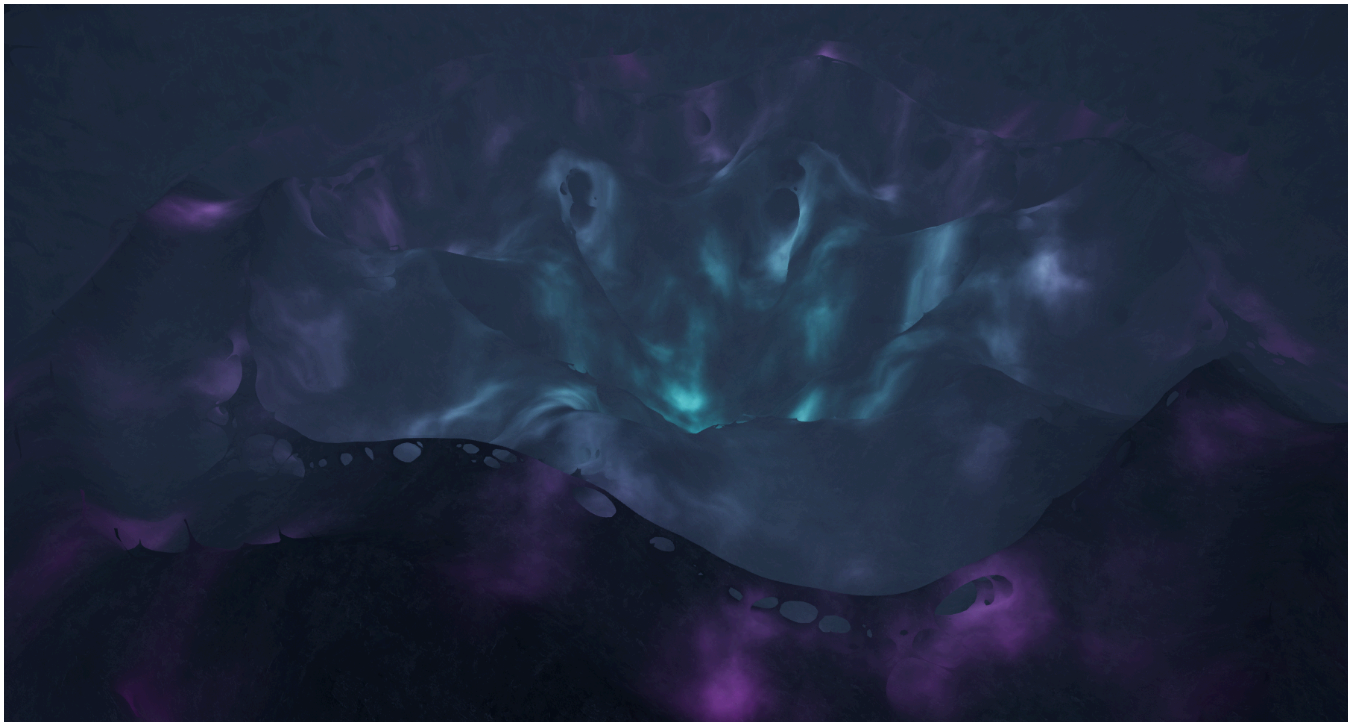
Here the resulting geometry can be seen:



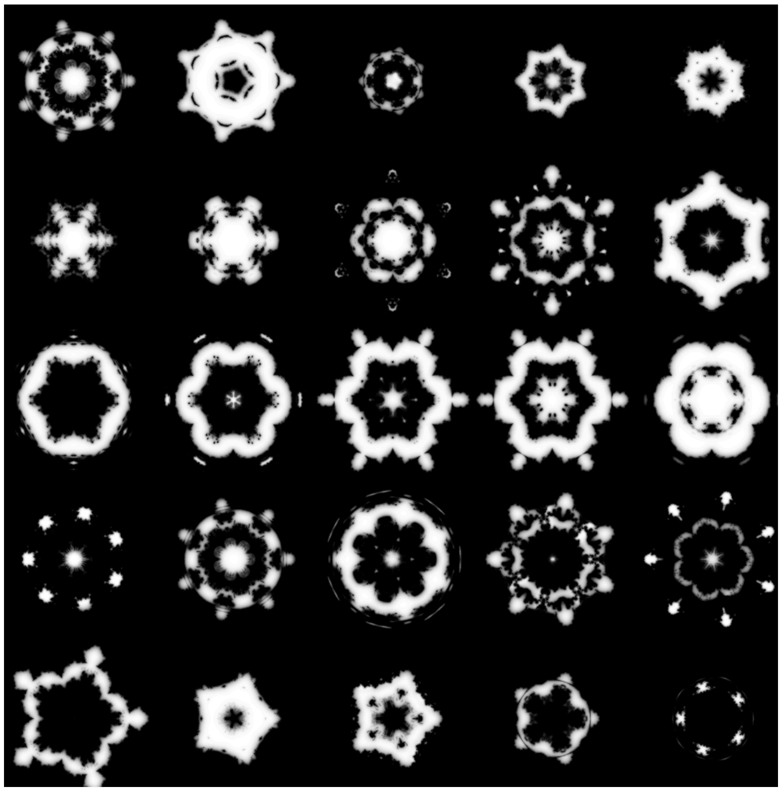
I was very pleased with the output - the ridges with the cobweb looking structures appeared other-worldly while still looking like terrain with smooth hills, ridges and erosion.
I took several sections of the mandelbulb to have some variation - I used a more messy section of the mandelbulb which resulted

To make this look like mountainous terrain, I used a Slate texture created by photographing some tiling outside of my house, and running it through Substance Sampler’s AI to Material filter to make it tile. Referring to my concept art for the scene, I wanted to add some extra flair to the centre of the landscape, so I created a simple shader that uses perlin and voronoi noise textures panning and multiplied through each other to create a flowing glow that lights up the centre of the landscape.

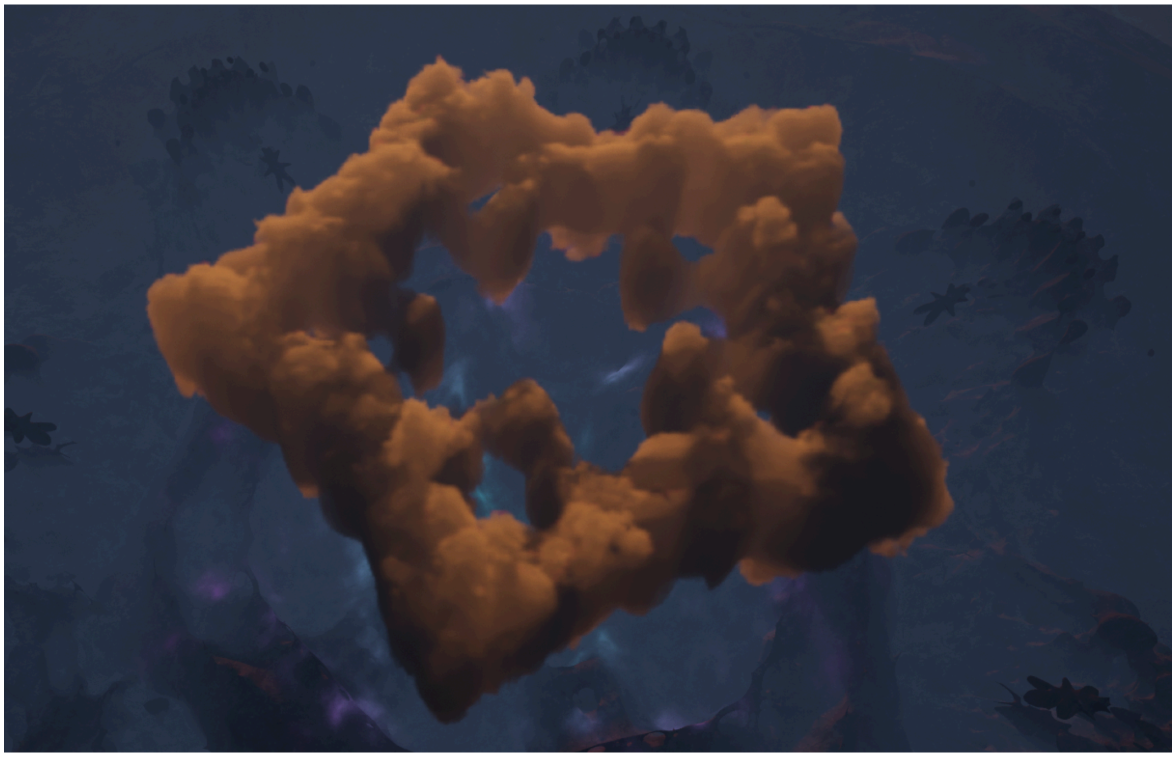




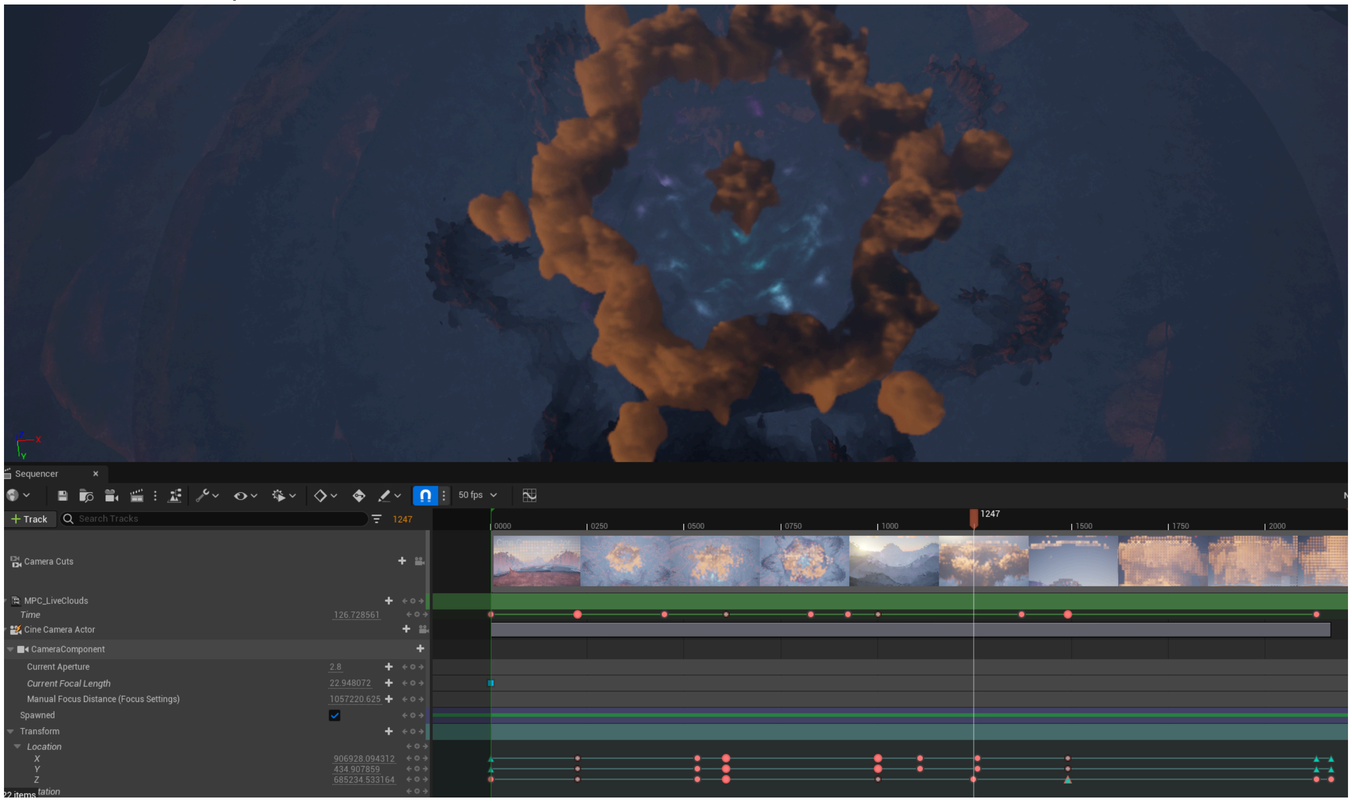
For the clouds, I wanted to morph through various shapes. Using photoshop, I cherry picked several frames from three volume textures, each with 6, 7 and 8 iterations respectively; producing different shapes.
I then composed these over smaller shapes to create even more diverse and interesting looking shapes, and collaged them into a volume texture. I then used gaussian blurs using different blending layers on top of this to create smooth fades on the edges which will negate this sharp edge effect as mentioned earlier.



This produced the exact result I wanted.



Modifying the master material to use a custom float value in place of Time, I was able to animate the sequence in which the clouds change through the shapes by using a Material Parameter Collection within a Level Sequencer.



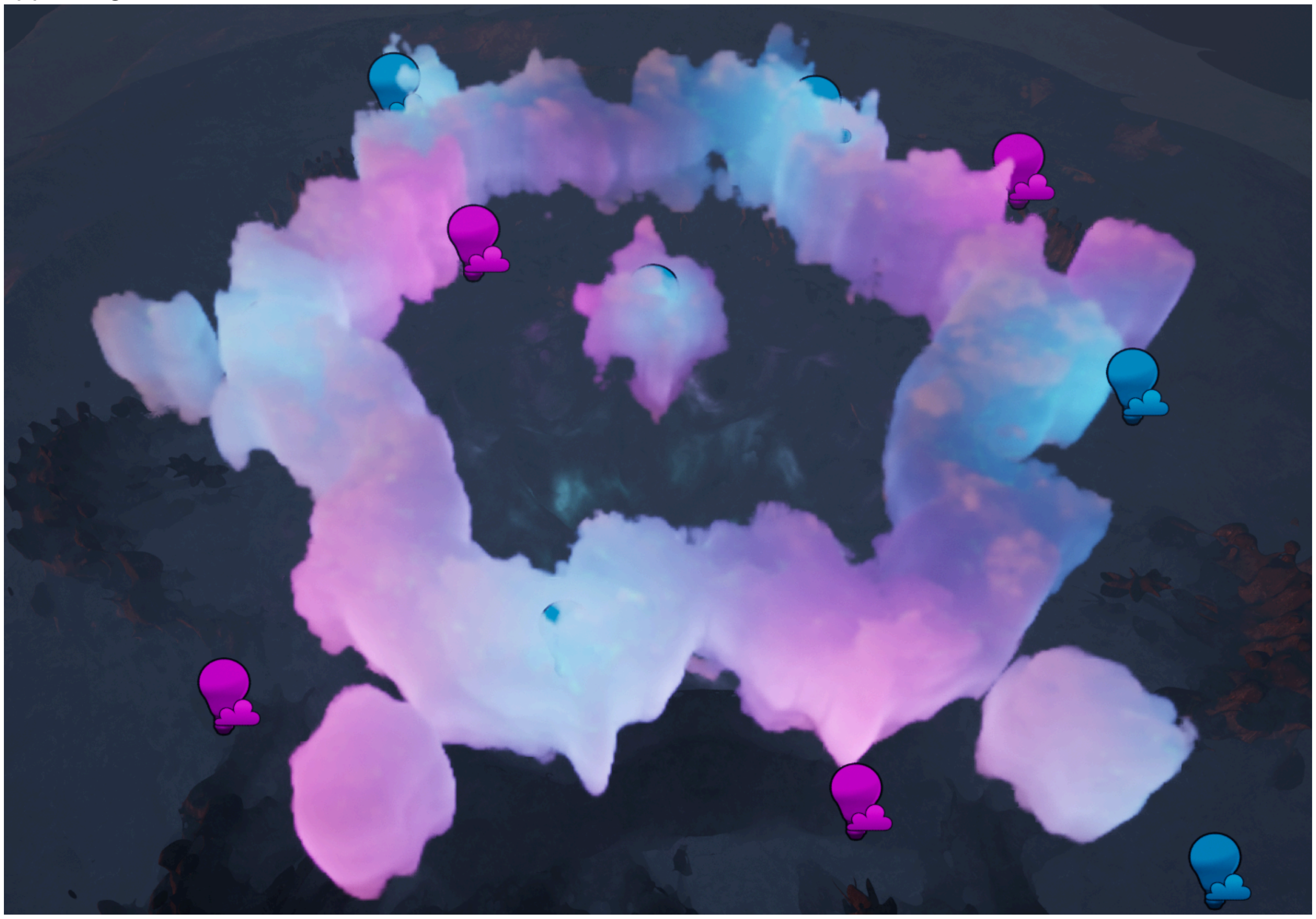
Sequencer

Track 1247

Camera Cuts
MPC_LiveClouds
Time 126.728561
Cine Camera Actor
CameraComponent
Current Aperture 2.8
Current Focal Length 22.948072
Manual Focus Distance (Focus Settings) 1057220.625
Spawned ✓
Transform
Location
X 906928.094312
Y 434.907859
Z 685234.533164

Although this looked great, I wanted to have the clouds have more colour, and fit into the environment more, so I scouted out for a plugin that would help me do this. I then found the ‘Clouds Lighting System’ plugin on the Unreal Engine Marketplace.

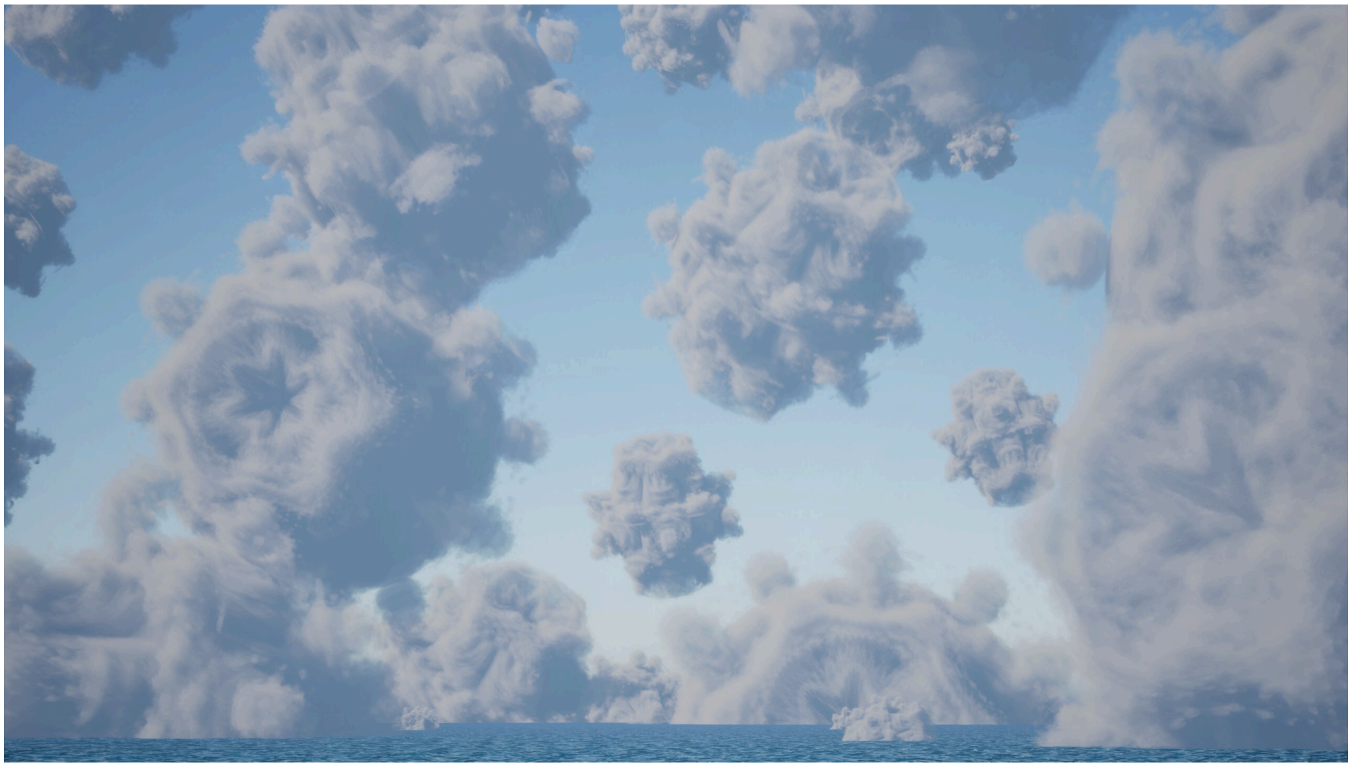
This plugin adds multiple light sources that are made for volumetric materials, which are fully customisable and accessible via blueprints. Originally intended for Unreal's 'Paint Clouds', I adapted the material functions for use within the master cloud material that I had been using. With this, I could easily add volume light sources that would only impact the clouds themselves, and gave a very appealing look.



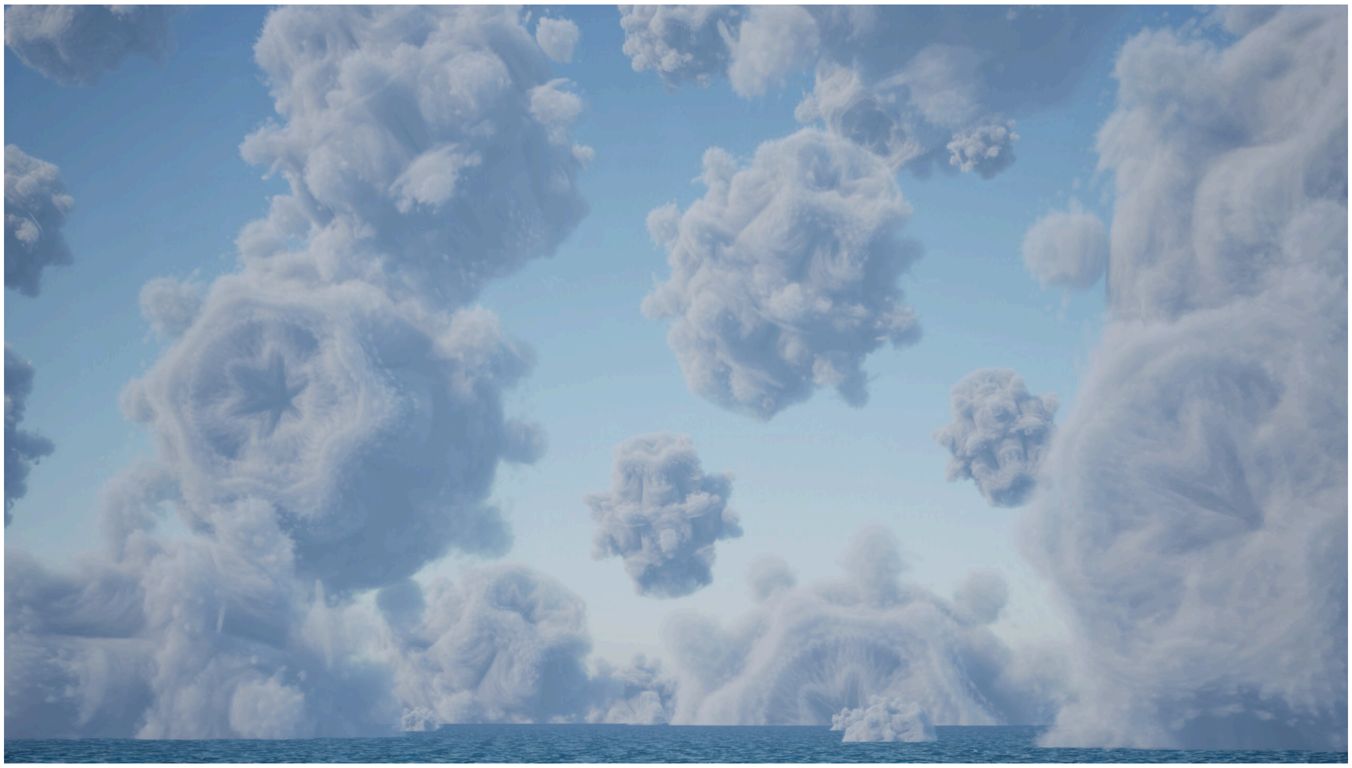
Additionally, multiplying the volume light material function by the cloud noise produces a more realistic look as it blends into the clouds rather than just lights up areas.

Sparse Volumetrics

I wanted to create a scene where an expansive blue ocean contained several "fallen clouds" of various sizes that stretch across the water, as well as this I wanted the scene to contain several more clouds floating in the sky that the camera would pan up to. Achieving the basic setup was pretty simple, hand placement of the VDB actors was done by taking into account where the camera will be throughout the scene, and the water was made using Unreal's default water plugin, with some of my own modifications to the scattering and absorption as well as the waves.
As mentioned earlier, using improved sampling and higher samples-per-pixel counts stops this scene from being run in real-time, so I intended to create two renders, one with the highest quality possible, and one with many render settings changed on the VDB actors, to allow them to run in real-time. My aim was to get both these renders as close as possible, getting to the point where I could have the VDB actors run in real-time as close to the highest quality with as small a sacrifice of quality as possible.



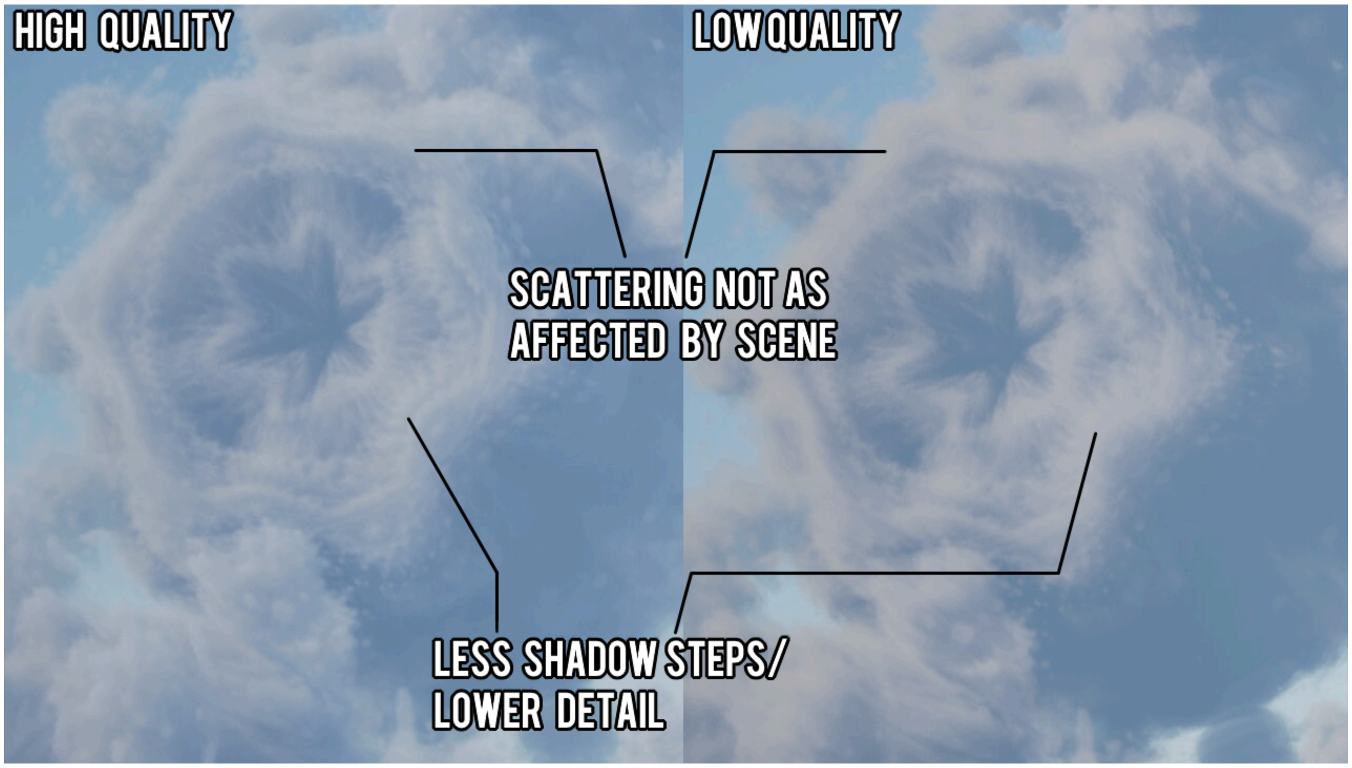
ABOVE: Lower Quality with tuned & optimised settings [90 FPS+ in real-time]
BELOW: Highest Quality Settings [3 FPS* in real-time]



*After experimenting with all the different features and options, I came to learn what each one did. So I went back to the original highest quality settings, and I disabled trilinear sampling interpolation, and

tweaked a couple of values by a small amount and I was able to reach around 20-30 FPS. Though I didn't feel that this range is an acceptable real-time FPS count as the standards for the absolute minimum in the gaming industry in 2023 is 60FPS, so this is what I aimed for.
Due to the fact that even tweaking these settings did not reach this goal I re-enabled them for the most detail possible for the high-quality render.

After much experimentation, I managed to get both results as close as possible, however there are definitely notable differences. The most is that the ambient scattering, largely affected by whether improved sunlight sampling is enabled, changes the highlights on the clouds that make it appear to not take in the scene's lighting as much. You would expect in this scene with a large blue ocean that it would appear more like the left side, less so of the right side where it appears more white as though it is being lit without taking in the surrounding scene as much. Additionally, to bring the FPS up to a


HIGH QUALITY
LOW QUALITY
SCATTERING NOT AS AFFECTED BY SCENE
LESS SHADOW STEPS/ LOWER DETAIL

Ultimately, I was able to produce a nice looking result that worked for real-time, however this was on a NVIDIA RTX 3080TI, one of the more recent graphics cards and therefore despite adjusted optimisation settings allowing real-time use on this graphics card, it is likely to be unsuitable for lower end graphics cards.

Conclusion

Ray marching is a practical approach for rendering volumetrics in Unreal Engine. Unreal Engine, for instance, employs multiple optimisation techniques, as well as including volume textures to make animated volumetric clouds achievable and practical for real-time use within games and cinematic purposes.
In the case of NanoVDB, it can produce visually appealing and high-quality results, however it was primarily designed as an experimental tool for playing with sparse volumes. While it can render realistic VDB clouds and other volumetric simulations, it lacks real-time potential for games due to trade-offs between quantisation, optimisation and quality. Although, despite being only a few years old, NanoVDB has much potential for cinematic uses within Unreal-Engine, and many technical artists have already begun demonstrating this by using programs such as EmberGen to create volumetric simulations for cinematic purposes.
As mentioned earlier within this documentation, NanoVDB is not natively supported by Unreal Engine, and thus this is currently a highly experimental workflow with many quirks and limitations. However I believe it is only a matter of time before we see native support and use within cinematic applications, given how fast rendering technology is advancing.

Sources

Below is a table of resources used to assist development throughout this project. More information available on request.

Source	Creator(s)	Role	Link
Fluid Ninja	Andras Ketzer	Volumetric Flow Materials	https://www.unrealengine.com/marketplace/en-US/product/fluidninja-vfx-tools
Shaderbits GDC Pack	Shaderbits	Density RayMarch	https://shaderbits.com/blog
Shaderbits GDC Pack Fix [4.22+]	Michał Kłoś	Density RayMarch	https://github.com/sp0lsh/UEShaderBits-GDC-Pack
NanoVDB	NVIDIA	Real-time GPU support for OpenVDB	https://developer.nvidia.com/nanovdb
Sparse Volumetrics [OpenVDB and NanoVDB in Unreal]	Thibault Lambert, Maxime Dupart	Unreal Engine implementation of NanoVDB	https://github.com/eidosmontreal/unreal-vdb
Entagma Fractal Set	Moritz Schwind, Manual Merkle	VEX implementation of Mandelbulb Equation	https://youtu.be/_mwJ7mIYRWg
Clouds Lighting System	Lukerrr	Additional Volumetric lighting for renders	https://www.unrealengine.com/marketplace/en-US/product/a54a45caa08e41bb97b1c19cebfe1093
ChatGPT 3.5 & 4	OpenAI	Used throughout the project	https://openai.com/blog/chatg

		to assist with solving problems as well as assisting with documentation	pt
ResearchGate	ResearchGate	Used for understanding Ray-Marching optimisation techniques such as adaptive sampling	https://www.researchgate.net/figure/Adaptive-sampling-a-with-a-volume-ray-marcher-and-b-with-a-particle-tracer-As-the_fig2_358820596
Art Hiteca	Art Hiteca	HLSL Mandelbulb	https://www.youtube.com/@ArtHiteca